



TM

freeBSDTM **JOURNAL**

Sept/Oct 2015

**Improving
MemGuard
Support** for UMA on FreeBSD

ZFS *and*
BOOT
environments

CloudABI Pure Capability-based Security for **UNIX**

Sometimes it's good to be redundant.



Now Available! 5-9s uptime from the creators of pfSense®

**High Availability SG-4860-1U
and High Availability SG-8860-1U features include:**

- Quad Core Intel® Atom™ C2558 2.4GHz or 8 Core Intel® Atom™ C2758 2.4 GHz with AES-NI
- Flexible Configuration - firewall, LAN or WAN router, VPN appliance, DHCP Server, DNS Server, multi-WAN, high availability, load balancing, reporting and monitoring
- Fully extendable with add-on software packages such as Snort®, Squid, SquidGuard, Suricata, and many more.
- Powered by pfSense Open Source software. No maintenance or upgrade fees.
- Create VPNs to the Amazon Cloud easily with our with Amazon® AWS™ Wizard.
- Get started with our included High Availability Configuration Guide.



Shop now at the official pfSense store or authorized partners worldwide.

<http://store.pfsense.org/Hardware/HA.aspx>

pfSense® is a registered trademark of Electric Sheep Fencing, LLC. Intel and Intel Atom are trademarks of Intel Corporation in the U.S. and/or other countries. Amazon AWS and Amazon are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries. Snort is a registered trademark of CISCO.



Table of Contents

Vol. 2, Issue No. 5

FreeBSD Journal

Sept/Oct 2015

4 CLOUD ABI

Pure Capability-based Security for UNIX

Learn how and why the author began working on a new runtime environment called CloudABI, and how you can build and run your own applications on it. **By Ed Schouten**

10



ZFS and BOOT Environments

Boot environments can save a lot of pain. They can also inflict surprises to old FreeBSD hands. **By Michael W. Lucas**

14



Improving MemGuard Support

for UMA on FreeBSD. MemGuard is effective for dynamic detection of memory errors and a very good tool for developing and testing. **By Luke Chang-Hsien Tsai**

20



FreeBSD Virtualization Options

FreeBSD has long played a key yet humble role in Unix application and operating system containment and is poised to provide compelling virtualization options for organizations of all sizes. **By Michael Dexter**

COLUMNS & DEPARTMENTS

3 Foundation Letter We've grown another digit!
By George Neville-Neil

22 Conference Report BSDCam 2015
This annual FreeBSD Developer Summit in Cambridge, England, took place at the University of Cambridge Computer Laboratory, August 17 to 19, 2015.
By Robert N. M. Watson

26 svn update This issue's column highlights some of the exciting updates expected in upcoming FreeBSD releases. *By Glen Barber*

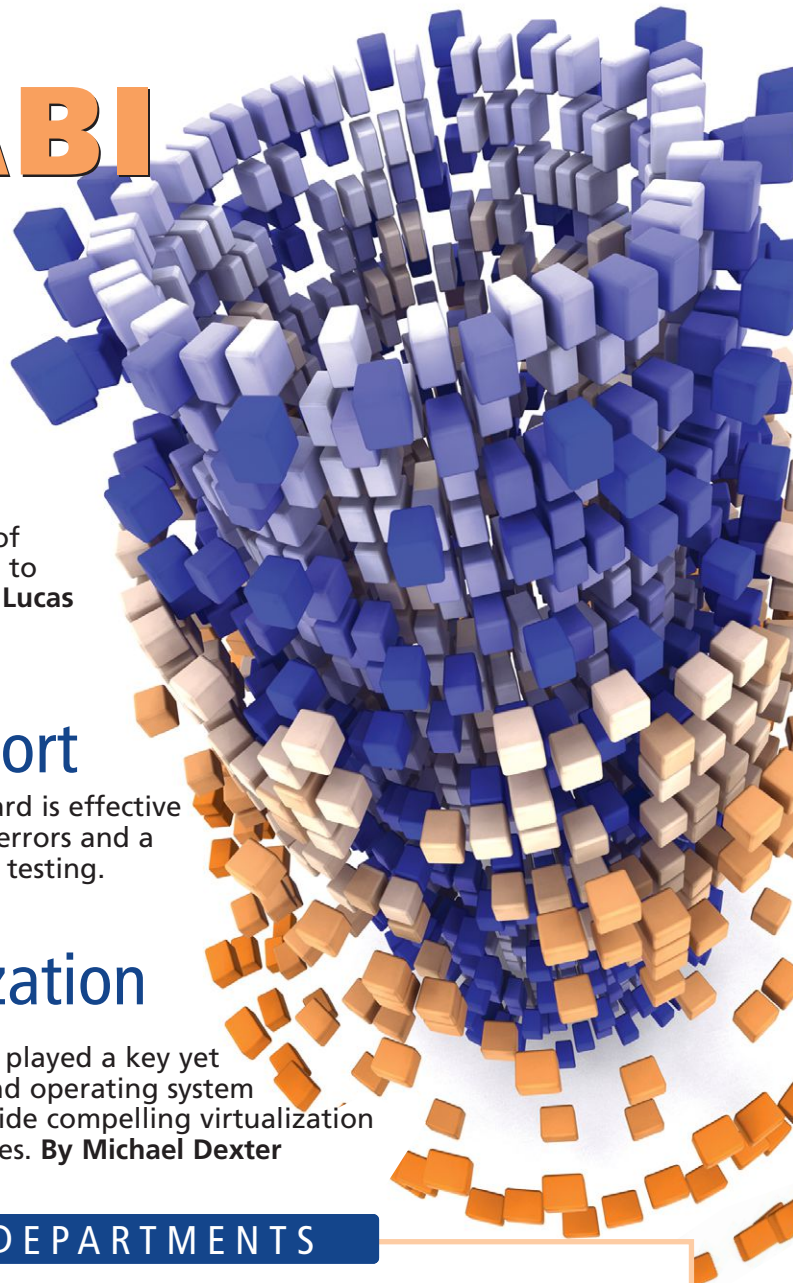
28 Book Review *The Practice of Cloud System Administration* by Thomas A. Limoncelli, Strata R. Chalup

and Christina J. Hogan. This is a presentation of current best practice that can help in every iteration of your design process. *By Greg Lehey*

32 Ports Report Although there was a decrease in activity of almost 25% in July and August, when considering the number of commits applied to the tree, major improvements were made. *By Frederic Culot*

34 This Month in FreeBSD The author interviews Warren Block, a longtime doc committer, member of the FreeBSD DocEng team, and the author of *igor*.
By Dru Lavigne

38 Events Calendar *By Dru Lavigne*



#MISSIONCOMPLETE



"CryptoLocker is a joke with ZFS"

Learn how Plextec defeats ransomware attacks with
FreeNAS and ZFS at ixsystems.com/cryptolocker

Have you used one of these tools to complete your mission?

Tell us more at ixsystems.com/missioncomplete for a chance to win monthly



#missioncomplete



FreeBSDTM JOURNAL

Editorial Board



- John Baldwin • Member of the FreeBSD Core Team
- Justin Gibbs • Founder and President of the FreeBSD Foundation and a senior software architect at Spectra Logic Corporation
- Daichi Goto • Director at BSD Consulting Inc. (Tokyo)
- Joseph Kong • Author of *FreeBSD Device Drivers*
- Dru Lavigne • Director of the FreeBSD Foundation and Chair of the BSD Certification Group
- Michael W. Lucas • Author of *Absolute FreeBSD*
- Kirk McKusick • Director of the FreeBSD Foundation and lead author of *The Design and Implementation* book series
- George Neville-Neil • Director of the FreeBSD Foundation and co-author of *The Design and Implementation of the FreeBSD Operating System*
- Hiroki Sato • Director of the FreeBSD Foundation, Chair of AsiaBSDCon, member of the FreeBSD Core Team and Assistant Professor at Tokyo Institute of Technology
- Robert Watson • Director of the FreeBSD Foundation, Founder of the TrustedBSD Project and Lecturer at the University of Cambridge

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

- Publisher** • Walter Andrzejewski
walter@freebsdjournal.com
- Editor-at-Large** • James Maurer
jmaurer@freebsdjournal.com
- Art Director** • Dianne M. Kischitz
dianne@freebsdjournal.com
- Office Administrator** • Michael Davis
davism@freebsdjournal.com
- Advertising Sales** • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
PO Box 20247, Boulder, CO 80308
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdjournal.org
Copyright © 2015 by FreeBSD Foundation.
All rights reserved.

This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER from the Board

Over 10,000 Subscribers!

We've grown another digit! That's right—the *FreeBSD Journal* now has more than 10,000 subscribers.

Those of you who also follow us on social media have already seen the announcement, but it's great to be able to put that down in the pages of the *Journal* itself. When we started the *Journal*, we really had no idea how many people we'd be able to reach, and we're extremely happy to have reached all of you. Of course this is no time for us to sit back and relax, which is why the editorial board and staff at the *Journal* are continuing to work to bring you all the best in the world of FreeBSD.

Our slate of articles this month ranges from deep within system's memory, with Luke Chang-Hsien Tsai's piece on MemGuard, over to a piece by Michael Lucas on booting with ZFS, out to Michal Dexter's FreeBSD Virtualization Options, and finally arriving in the cloud with Ed Schouten's article on CloudABI. Of course, all our columnists are back—Glen Barber keeps us connected to what's new in the source tree with "svn update," Frederic Culot lets us know how the ports are doing, and Dru Lavigne not only updates us with the Events Calendar but also provides us with more history in "This Month in FreeBSD."

It seems like there were more FreeBSD and BSD-related conferences than ever this year—so many that I can't even make it to all of them anymore. That's great news because it shows the growth of our community. If you weren't able to make it to the BSDCam Developer Summit at Cambridge University this year, you can at least get a taste of what went on by reading Robert Watson's Conference Report. Developer summits are smaller than conferences, and have a great collegiate feel, where smaller groups of people can make plans and build prototypes of the next things we're going to put into FreeBSD.

Our issue rounds out with a book review, where Greg Lehey, a longtime contributor to FreeBSD, reviews *The Practice of Cloud System Administration*, by Thomas Limoncelli, Strata Chalup, and Christina Hogan.

We have one more issue to put out in 2015, but we're already planning a slate of new articles for 2016, and we're looking forward to reaching even more folks next year.

Sincerely, 
George Neville-Neil

For the *FreeBSD Journal* Editorial Board

CloudABI

PURE capability-based security
for **UNIX**

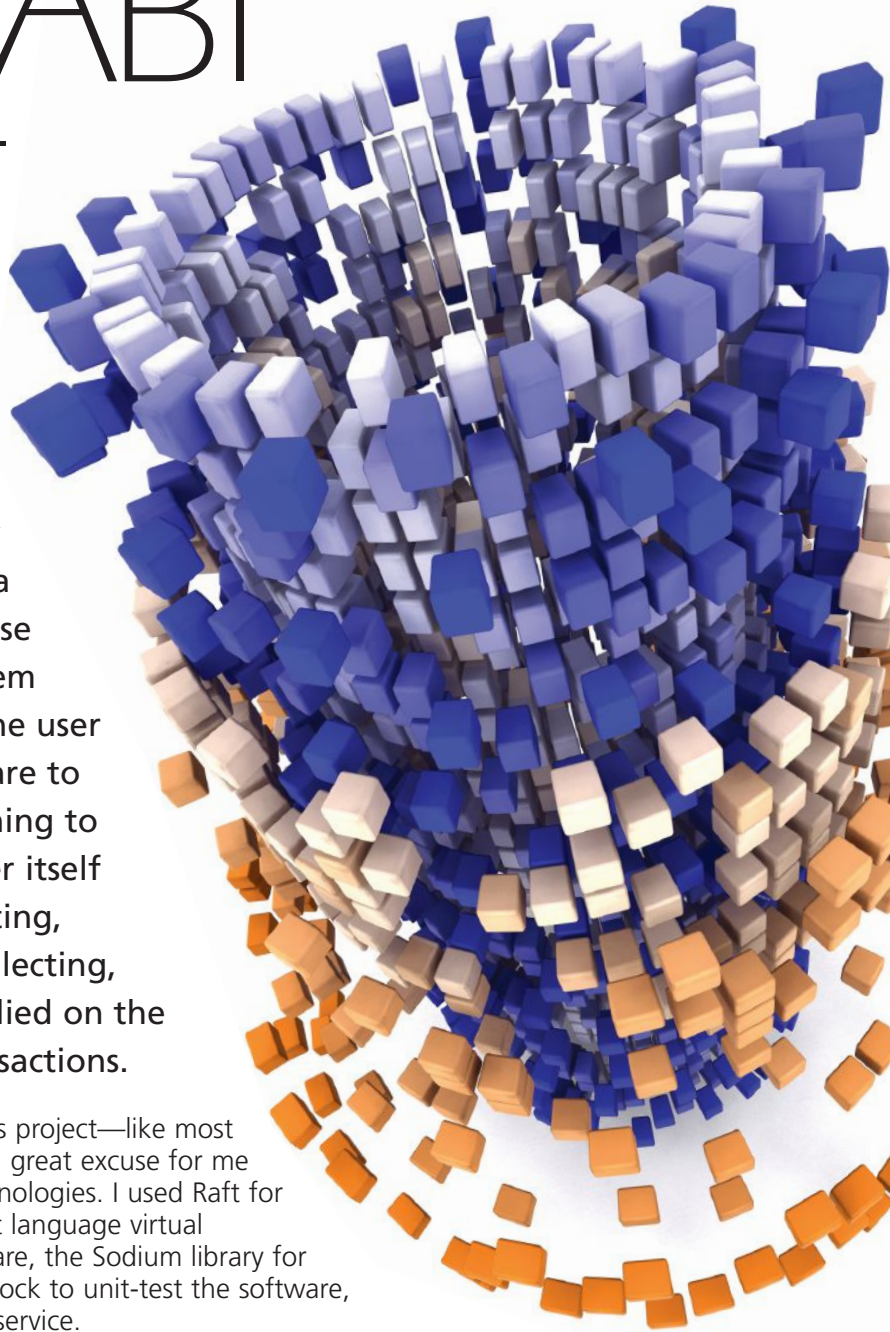
by Ed Schouten

Late last year I was writing a distributed storage service in my spare time. Instead of it having a fixed schema (like how a database server uses tables and a file system uses a tree), the idea was that the user would provide a piece of firmware to actually give structure and meaning to the data stored inside. The server itself was capable of securely distributing, caching, replicating, garbage collecting, and repairing the data, but it relied on the firmware to perform actual transactions.

Although I eventually abandoned this project—like most of my spare-time projects—it was a great excuse for me to experiment with some new technologies. I used Raft for Paxos-like distributed consensus, the Dart language virtual machine to parse and execute the firmware, the Sodium library for cryptography, Google Test and Google Mock to unit-test the software, and FreeBSD’s Capsicum to sandbox the service.

While I was implementing all of this, I really started to enjoy using Capsicum. Access control-based security frameworks like AppArmor tend to push the burden of making software run securely towards the user, who may not necessarily have knowledge of all of the internals of the application. If an AppArmor policy does not work, typically an average user has no choice but to disable it. Capsicum solves this problem by integrating the security policy into the design of the application itself. Applications become safe without requiring any additional configuration. Nice and simple.

What I also like about Capsicum is that it guides you to write testable software. As there are no longer any global namespaces, all interaction with the system needs to be performed through handles, namely file descriptors. It is no longer possible to access files through hardcoded pathnames, nor is it possible to open connections to hardcoded hosts on the network. You could argue that Capsicum forces you to use dependency injection, which is a good thing, in my opinion.



Does Capsicum Scale?

There is one thing that bothered me about Capsicum, though. What I've noticed is that Capsicum *scales worse than linear*. With that I mean that it seems to become harder and harder to modify applications to work correctly with Capsicum as they become larger. Once you sandbox a process, you instruct the kernel to disable a very large number of system calls. For example, all file system-related system calls except for `*at()` suddenly return `ENOTCAPABLE`. All code that either directly or indirectly invokes these system calls behaves differently as a result of that. In applications that use a large number of libraries, it eventually becomes impossible to determine which functions are still safe to use after sandboxing. This caused a lot of churn in my case when I tried to make the Dart virtual machine work in a sandbox.

I noticed that these problems can already be observed when using FreeBSD's core libraries. It is no longer possible to create locale objects after sandboxing, meaning you can't parse strings in a user-provided character set. Another example is that it becomes impossible to access the time zone database after sandboxing. This means that your application can only use UTC and the system-wide time zone, assuming it already got loaded into memory before sandboxing.

The worst example that I encountered was in a cryptography library called libtomcrypt that I installed from the Ports tree. This library will only use strong entropy for its random number generator outside of capabilities mode. It will silently fall back to using the system clock as its sole source of entropy after entering capabilities mode, as it can no longer access `/dev/urandom`.

CloudABI: Pure Capability-based Computing

To make it easier to write software that works well in such a sandboxed environment, I started working on a new runtime environment called CloudABI. Where CloudABI differs from FreeBSD's standard runtime is that processes already start up in capabilities mode. This allows us to remove all system calls that are incompatible with Capsicum. The impact of this is that any code that uses these incompatible features becomes very easy to detect: it will simply fail to build. Though this may sound radical at first, my experience is that this actually saves a lot of time. Fixing build failures is often a lot simpler than trying to track down regressions caused by enabling Capsicum in the standard runtime.

Another interesting aspect of having a pure capability-based runtime is that it becomes possible to run untrusted third-party applications directly on top of the kernel without requiring any complex access control lists, security policies, virtual machines, or containers. Access can be fully controlled by adjusting permissions on file descriptors prior to execution. It allows you to create "tiny computing environments" in which processes can only communicate with the environment through a restricted set of RPCs. This makes it an interesting building block for secure microservices architectures, hence the name CloudABI.

Once you remove all system calls that either conflict with Capsicum, are implemented only for legacy reasons, perform administrative tasks, are otherwise privileged, or are redundant, the UNIX ABI becomes very compact. Whereas FreeBSD has 338 system calls that are enabled by default, CloudABI only has 58. This makes it easy to add support for CloudABI to existing operating system kernels, allowing you to run the same executable on multiple systems, without requiring any recompilation.

CloudABI has already been fully ported to FreeBSD and NetBSD. These ports are both only about 5,000 lines of code in size. An experimental Linux port is also available.

Building Our Very First CloudABI Program

Now that we know what CloudABI is, it's time get our hands dirty by taking a look at how we can build and execute our very own CloudABI programs. Let's start off by building a simple "Hello, world" application. We'll see that even such a simple application provides a lot of insight into how CloudABI works in practice.

As CloudABI's runtime is separate from FreeBSD's native programming environment, the first step is to install a cross compiler toolchain that we can use to generate CloudABI ELF executables. To accomplish this, we can just install the `cloudabi-toolchain` package from the FreeBSD ports tree. This package installs an unmodified copy of LLVM 3.7 and creates a symbolic link from `x86_64-unknown-cloudabi-cc` to the Clang executable. Clang is smart enough to automatically detect that it is being invoked as a cross compiler for CloudABI when it is invoked through this symbolic link. It also installs a copy of

GNU Binutils targeted against CloudABI.

Right now the `cloudabi-toolchain` package only installs a cross compiler for x86-64. This package will be extended to set up additional symbolic links once CloudABI is ported to more hardware platforms, so that it becomes easy to build software for any architecture.

```
$ sudo pkg install cloudabi-toolchain
$ cat hello.c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello, world\n");
    return 0;
}
$ x86_64-unknown-cloudabi-cc -o hello hello.c
hello.c:1:10: fatal error: 'stdio.h' file not found
```

Once we install the toolchain package and try to compile our simple program, we'll observe that the build still fails, as the compiler cannot find the `<stdio.h>` header. The reason for this is that the CloudABI toolchain in the FreeBSD ports tree does not ship with any libraries built for CloudABI. It merely contains the development tools that you'd want to run on your FreeBSD system. But fear not; there is no need for us to

cross-build any of the CloudABI core libraries manually.

The CloudABI project has its own ports collection, simply called CloudABI Ports. The idea behind having a separate ports collection for CloudABI is that it allows us to generate packages not just for FreeBSD, but also for the other BSDs and various Linux distributions. The packages contain exactly the same cross-compiled binaries and libraries, but are packaged in such a way that they can be installed and upgraded by the native package manager on your system. Even if your operating system does not run CloudABI executables, there should be no reason why you can't develop software on it.

Instructions on how to add the CloudABI package repository to `pkg`'s configuration can be found on the CloudABI Ports GitHub page (<https://github.com/NuxiNL/cloudabi-ports>). We won't describe this process here, as we want to spare you from typing over the PEM file that is used to verify packages. Once you've added the repository, you should be able to install the `x86_64-unknown-cloudabi-c-runtime` package to obtain a set of basic libraries for C programming, such as CloudABI's C library. For C++ development, you can install the `x86_64-unknown-cloudabi-cxx-runtime` package.

```
$ sudo pkg update
$ sudo pkg install x86_64-unknown-cloudabi-c-runtime
$ x86_64-unknown-cloudabi-cc -o hello hello.c
hello.c:3:3: warning: implicitly declaring library function
'printf' with type 'int (const char *, ...)'
hello.c:(.text.main+0x1c): undefined reference to `printf'
```

After installing the C runtime, the compiler will be able to find `<stdio.h>`. The build should now continue, but we still get a nasty linker error, as it cannot find the `printf()` function. As we'll see later on, file descriptors 0, 1, and 2 don't have a fixed purpose, which

is why `stdout` and its helper functions are not present. For now, let's assume that we'll be running our application directly from the shell, where file descriptor 1 corresponds to the TTY. We can use the `dprintf()` function to print to this file descriptor directly.

```
$ cat hello.c
#include <stdio.h>
int main(int argc, char *argv[]) {
    dprintf(1, "Hello, world\n");
    return 0;
}
$ x86_64-unknown-cloudabi-cc -o hello hello.c
```

Now that we managed to get our code to build, let's go ahead and run our program. To be able to do that, we need to make sure that we load a pair of kernel modules. These have been fully imported into the FreeBSD source tree as of svn revision 286680, meaning that any snapshot of FreeBSD 11.0-CURRENT from August 13, 2015 or later ships with these kernel modules by default.

Approximately 85% of the CloudABI system calls are designed in such a way that they behave identically across hardware architectures. The constants, data types, and structures that they depend on are exactly the same. These machine independent system calls are provided by the `cloudabi` kernel module.

For the system calls that do depend on hardware specific properties, such as pointers being 64-bits in size, we provide a separate kernel module called `cloudabi64`. This kernel module also provides the code that is necessary to load the 64-bit ELF executables into memory and begin their execution.

The advantage of having this separation is that if CloudABI were to gain support for a 32-bit architecture as well, we can reuse the implementations of all of the architecture-independent system calls. The `cloudabi32` kernel module would only need to implement a fraction of the ABI.


```
$ ./hello
ELF binary type "17" not known.
$ sudo kldload cloudabi64
$ kldstat
Id Refs Address          Size      Name
  1     4 0xffffffff80200000 1e7f940 kernel
  2     1 0xffffffff82221000 5a61      cloudabi64.ko
  3     1 0xffffffff82227000 b167      cloudabi.ko
$ ./hello
Hello, world
```

Starting Programs Safely with **cloudabi-run**

One of the implications of a purely capability-based runtime environment is that it becomes very important that processes are started with the correct set of file descriptors. We need to have exact control of which file descriptors are available to the new process. Unused file descriptors that are leaked into the

new process by accident will grant more rights to the process than strictly required. On UNIX-like systems, this may normally be controlled through close-on-exec flags that are set on file descriptors, but it is well known that these are hardly ever set to the right value.

As a program becomes more complex, it also gets harder to launch it with the right set of file descriptors. For example, if a web server can only listen on a single network socket and serve files stored in a single directory, it is still possible to use a fixed file descriptor table layout. Once the web server gains support for listening on multiple sockets, can serve files for multiple virtual hosts, and can send queries to a redundant pool of database backends, it becomes hard to launch the process correctly. The process somehow needs to be informed about the layout of the file descriptor table.

To solve this, CloudABI comes with a utility called **cloudabi-run(1)**. In a nutshell, this tool parses a configuration file written in YAML and passes its contents on to the CloudABI program. The configuration file may contain special tags that instruct **cloudabi-run** to substitute entries with file descriptors to sockets, files, or directories. Only those file descriptors are passed on to the program. The process can obtain the file descriptor numbers by traversing the configuration data. This means that programs don't need to make assumptions about file descriptor numbers being reserved for a certain purpose. The YAML data replaces the conventional command line arguments and can be accessed from within the program

RootBSD

Premier VPS Hosting

RootBSD has multiple datacenter locations,
and offers friendly, knowledgeable support staff.
Starting at just \$20/mo you are granted access to the latest
FreeBSD, full Root Access, and Private Cloud options.



www.rootbsd.net

by using the alternative entry point `program_main()`.

Let's see how `cloudabi-run` works in practice by creating a tiny web server. The configuration file for this web server uses a very simple format: a list of key-value pairs. Right now we only support a single configuration entry called "socket", which should hold a file descriptor to a network socket. The goal is to eventually support more configuration options, such as a "rootdir" option to provide a file descriptor to the root directory of the web server. Making that work is left as a homework exercise.

After the program starts up through `program_main()`, we iterate through all of the key-value pairs using `argdata_iterate_map()`. The `parse_config()` callback then extracts the file descriptor number from the "socket" entry. Once we've obtained our network socket, we repeatedly call `accept()` on this socket to respond to incoming network connections. To keep this example simple, we just return a fixed response to the client.

In our configuration file we use a special tag, `!socket`, to instruct `cloudabi-run` to create a network socket that is listening on all IPv4 addresses, using TCP port 12345. It also demonstrates how we can provide access to files and directories by using the `!file` tag. If we run `procstat -f` after spawning our web server, we can confirm that only the file descriptors listed in our configuration file (and the file descriptor that is in the process of being created by `accept()`) are present in our process.

What is nice about this example is that it shows how much startup code disappears from the program when we use `cloudabi-run`. There is no need for us to write a configuration file parser, as

```
$ cat webserver.c
#include <sys/socket.h>

#include <argdata.h>
#include <program.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

static bool parse_config(
    const argdata_t *key, const argdata_t *value,
    void *sockfd) {
    const char *keystr;
    if (argdata_get_str_c(key, &keystr) == 0 &&
        strcmp(keystr, "socket") == 0) {
        argdata_get_fd(value, sockfd);
    }
    return true;
}

void program_main(const argdata_t *ad) {
    int sockfd = -1;
    argdata_iterate_map(ad, parse_config, &sockfd);
    int connfd;
    while ((connfd = accept(sockfd, NULL, NULL)) >= 0) {
        /* TODO: Parse the request. Serve data from disk. */
        const char buf[] = "HTTP/1.1 200 OK\r\n"
                           "Content-Type: text/plain\r\n\r\n"
                           "Content-Length: 13\r\n\r\n"
                           "Hello, world\n";
        write(connfd, buf, sizeof(buf) - 1);
        close(connfd);
    }
    exit(1);
}

$ x86_64-unknown-cloudabi-cc -o webserver webserver.c
```

CODE BOX CONTINUES NEXT PAGE


```

$ cat config.yaml
%TAG ! tag:nuxi.nl,2015:cloudabi/
---
hostname: mywebserver.com
rootdir: !file
  path: /var/www/mywebserver.com
socket: !socket
  bind: 0.0.0.0:12345
logfile: !fd stdout
$ pkg install cloudabi-utils
$ cloudabi-run webserver < config.yaml &
[1] 869
$ procstat -f 869
PID COMM      FD T ... PRO NAME
...
869 webserver  0 v ... -   /var/www/mywebserver.com
869 webserver  1 s ... TCP 0.0.0.0:12345 0.0.0.0:0
869 webserver  2 v ... -   /dev/pts/0
869 webserver  3 ? ... -   -
$ fetch -go - http://127.0.0.1:12345/
Hello, world

```


`cloudabi-run` already does all of that hard work for us. There is no need for the web server to create a network socket either.

In our example we bind our web server to an IPv4 address, but `cloudabi-run` also allows you to use IPv6 or UNIX sockets, without requiring any modification to the web server. A UNIX socket may be useful if you want to test the web server. If we ever want to create sockets with custom parameters (e.g., different timeout and retransmission policy, binding to specific addresses), this functionality can all be added to `cloudabi-run`, meaning that any application launched through `cloudabi-run` gains that functionality without requiring any code changes.

Closing Words

Now that we've seen how you can build and run your own applications on CloudABI, be sure to explore the CloudABI Ports repository to see which pieces of software have already been ported. There are a lot of libraries that you can already use to create some pretty complex applications, but I'm sure you can think of packages that are still missing. It goes without saying that any contributions to CloudABI Ports (and any of the other CloudABI components) are welcomed.

- LINKS**
- CloudABI on GitHub: <https://github.com/NuxiNL/cloudlibc>
 - CloudABI Ports Collection on GitHub: <https://github.com/NuxiNL/cloudabi-ports>

 **ED SCHOUTEN** has been a developer at the FreeBSD Project since 2008. His contributions include FreeBSD 8's SMP-safe TTY layer, the initial import of Clang into FreeBSD 9, and the initial version of the vt(4) console driver that eventually made its way into FreeBSD 10.

CloudABI has been developed by the author's company, Nuxi, based in the Netherlands. It will always be available as open-source software, free of charge. Nuxi offers commercial support, consulting, and training for CloudABI. If you are interested in using CloudABI in any of your products, be sure to get in touch with Nuxi at info@nuxi.nl.

ZFS AND BOOT ENVIRONMENTS

Upgrading a host's operating system or its packages offers all sorts of risks. Maybe the new kernel will expose a subtle bug in your NFS clients, or the new web server version will choke on your PHP code. Remediating these problems requires a lot of experience, careful planning, debugging skills, and a certain amount of luck. A failed upgrade leads to that most difficult of questions: fix the problem, or roll back? Either one promises a whole bunch of work, probably when you need to do other things.

As of FreeBSD 10.1, new ZFS installs are specifically designed to support boot environments. Long a feature of Solaris-based systems, boot environments let you install multiple versions of the core operating system and choose which to boot into. If you create a boot environment every time you change your installed system, you have an easy path for reverting the change wholesale. Boot environments change other aspects of systems administration, though, and if you deploy them blindly you'll cause more trouble.

by
Michael W. Lucas

Dataset Layout

Boot environments are built on top of ZFS. Everything that should be included in the boot environment needs to be on a single dataset. Take a look at a new FreeBSD 10.1/amd64 install to see what I mean.

```
# zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
zroot	465M	188G	128K	none
zroot/ROOT	463M	188G	128K	none
zroot/ROOT/default	462M	188G	462M	/
zroot/tmp	149K	188G	149K	/tmp
zroot/usr	570K	188G	128K	/usr
zroot/usr/home	186K	188G	186K	/usr/home
zroot/usr/ports	128K	188G	128K	/usr/ports
zroot/usr/src	128K	188G	128K	/usr/src
zroot/var	703K	188G	128K	/var
zroot/var/crash	128K	188G	128K	/var/crash
zroot/var/log	192K	188G	192K	/var/log
zroot/var/mail	128K	188G	128K	/var/mail
zroot/var/tmp	128K	188G	128K	/var/tmp

This ZFS install has created separate datasets for /usr, /var, as well as subdirectories like /var/log and /usr/ports. The list is deceptive, though. Compare the list of datasets to the output of mount(8).

```
# zfs list
# mount
zroot/ROOT/default on / (zfs, local, noatime, nfsv4acl)
devfs on /dev (devfs, local, multilabel)
zroot/tmp on /tmp (zfs, local, noatime, nosuid, nfsv4acl)
zroot/usr/home on /usr/home (zfs, local, noatime, nfsv4acl)
zroot/usr/ports on /usr/ports (zfs, local, noatime, nosuid, nfsv4acl)
zroot/usr/src on /usr/src (zfs, local, noatime, nfsv4acl)
zroot/var/crash on /var/crash (zfs, local, noatime, noexec, nosuid, nfsv4acl)
zroot/var/log on /var/log (zfs, local, noatime, noexec, nosuid, nfsv4acl)
zroot/var/mail on /var/mail (zfs, local, nfsv4acl)
zroot/var/tmp on /var/tmp (zfs, local, noatime, nosuid, nfsv4acl)
zroot/var/www on /var/www (zfs, local, noatime, nfsv4acl)
```

Not all of the existing datasets are mounted! A quick check shows that /var and /usr are not mounted, although child datasets like /usr/ports and /var/log are. What's going on?

The /var and /usr datasets exist solely as parent datasets for children like /usr/home and /var/crash. Files under /usr/home/ exist on their own dataset, while files under directories like /usr/bin, /usr/sbin, and /var/db are actually on the root dataset. This is highly counterintuitive to those of us accus-

tomed to more traditional files systems. ZFS not only lets it happen—it uses this as a feature.

Whether you're upgrading your base system or packages, a few directories on FreeBSD are absolutely vital. FreeBSD's core program binaries lurk in /bin, /sbin, /usr/bin, and /usr/sbin. Thanks

to the non-mounting /usr dataset, these directories are now on your root dataset. Packages live under /usr/local—but that's also on your root dataset. The /var/db directory contains the package database, records for freebsd-update, and even the locate database. Because /var isn't mounted, these critical /var/db files are now part of the root dataset.

Files that are not part of the core system—such as the logs, user home directories, and more—are their own datasets.

You can now manage the operating system and add-on packages as a single entity.

Boot Environments and ZFS

One of ZFS's features is the powerful snapshot facility. Not only can you snapshot a dataset, but you can roll back the filesystem to that state. You can clone a snapshot into a new live copy of the file system.

This has obvious applications to systems administration. Take a snap-

shot of your system before running your upgrade. If the upgrade fails, roll back to the known working version. To debug a failed upgrade, copy the snapshot of the failed upgrade off to a test system and debug it there, while your production system keeps chugging along on the slightly older operating system version.

"Boot Environments" are a way of packaging all of this up in a nice neat bundle and putting a ribbon around it. You don't need a boot environ-

ment manager to use boot environments, but they make it much easier. Solaris-based systems use beadm(8), or boot environment administration, to handle boot environments. FreeBSD has an add-on beadm package. It's deliberately designed to resemble the Solaris program—while we could have created a new FreeBSD-specific tool, there's really no reason to.

Installing beadm

Use pkg(8) to get beadm. Here I'm installing beadm on a brand-new 10.1 machine.

```
# pkg install -y beadm
```

Now see which boot environments you have.

```
# beadm list
```

BE	Active	Mountpoint	Space	Created
default	NR	/	494.0M	2015-04-08 07:18

The only boot environment is named default. Under active, N means the environment is active now. An R means the environment will be active on reboot. So, the default environment is active, and will be active again after the next reboot.

This makes perfect sense. This machine has only one boot environment—I just installed it.

I need to upgrade this host to the latest version of FreeBSD 10.1, p9. This upgrade could go wrong—it probably won't but it could. I want a new boot environment for it. I'm going to call this environment install, because I just installed.

```
# beadm create install
Created successfully
# beadm list
```

BE	Active	Mountpoint	Space	Created
default	NR	/	646.0M	2015-04-08 07:18
install	-	-	10.7K	2015-04-08 11:43

The "install" boot environment is a read-only snapshot of my existing root dataset. Should I have problems, I can reactivate this version. The "default" boot environment is the currently live root dataset.

Before I run any upgrade or perform any major system changes, such as updating the installed packages, I create a new boot environment that reflects the state of the current system.

Each of these represents an upgrade that could have gone wrong.

```
# beadm list
```

BE	Active	Mountpoint	Space	Created
default	NR	/	3.2G	2015-04-28 11:53
install	-	-	126.0M	2015-04-28 12:19
10.1-p9	-	-	209.0M	2015-05-14 08:01
10.1-p10	-	-	166.0M	2015-05-24 11:02

Falling Back

If an upgrade goes bad, activate the last known good boot environment with "beadm activate" and the boot environment name.

```
# beadm activate 10.1-p10
Activated successfully
# beadm list
# beadm list
```

BE	Active	Mountpoint	Space	Created
default	N	/	3.2G	2015-04-28 11:53
install	-	-	126.0M	2015-04-28 12:19
10.1-p9	-	-	209.0M	2015-05-14 08:01
10.1-p10	R	-	166.0M	2015-05-24 11:02

The environment "default" has an activate of N, indicating that it's now running. The "10.1-p10" environment has an R, showing that it will be active after a reboot.

Falling back from a failed upgrade is now trivial.

Behind the Scenes

```
# zfs list -r zroot/ROOT
```

NAME	USED	AVAIL	REFER	MOUNTPPOINT
zroot/ROOT	3.21G	26.9G	96K	none
zroot/ROOT/10.1-p10	12K	26.9G	2.19G	/
zroot/ROOT/10.1-p9	12K	26.9G	2.12G	/
zroot/ROOT/default	3.21G	26.9G	2.51G	/
zroot/ROOT/install	8K	26.9G	612M	/

What does a boot environment do at the ZFS level? Take a look at the datasets under our root. We now have several datasets under zroot/ROOT. If you check their origins (with “zfs get -r origin zroot/ROOT”), you’ll see that each is a clone of the default dataset at a specific date and time.

If you want to see what’s in an older boot environment, check the source snapshot. If you want to change an unused boot environment, mount it at a temporary location.

Boot Environments and Common FreeBSD Practices

Boot environments can save a lot of pain. They can also inflict surprises to old FreeBSD hands.

The biggest mental shift you’ll need to make is to remember that directories like /var/db and /usr/local are under the boot environment. If you fall back to an older boot environment, those directories revert with it.

FreeBSD’s MySQL package keeps its database files in /var/db/mysql. If you fall back to an older boot environment, you’ll suffer a database reversion. The PostgreSQL package stores files in /usr/local/pgsql. Changing boot environments takes data with it. Many web server programs stuff files in /usr/local/something.

The list goes on.

Databases and applications really need their own ZFS datasets for their data. I create zroot/var/mysql, zroot/var/www, and so on, and tell the applications to use these datasets. These are easily configured via rc.conf. This rc.conf entry tells the mysql package to put its data in /var/mysql.

```
mysql_dbdir="/var/mysql"
```

This is a very minor change for a powerful tool that can save you huge amounts of pain, though. ●

Michael W. Lucas is the author of *Absolute FreeBSD*, *Absolute OpenBSD*, and *DNSSEC Mastery*, among others. He lives in Detroit, Michigan, with his wife and a whole mess of rats.

Visit his website at <https://www.michaelwlucas.com>.



FreeBSD JOURNAL

NOW AVAILABLE AS A DYNAMIC EDITION!

The Dynamic Edition format offers subscribers the same features as the App, but permits viewing the Journal through your favorite browser.

www.freebsdoundation.org

The DE, like the App, is an individual product. You will get an email notification each time an issue is released.

\$19.99

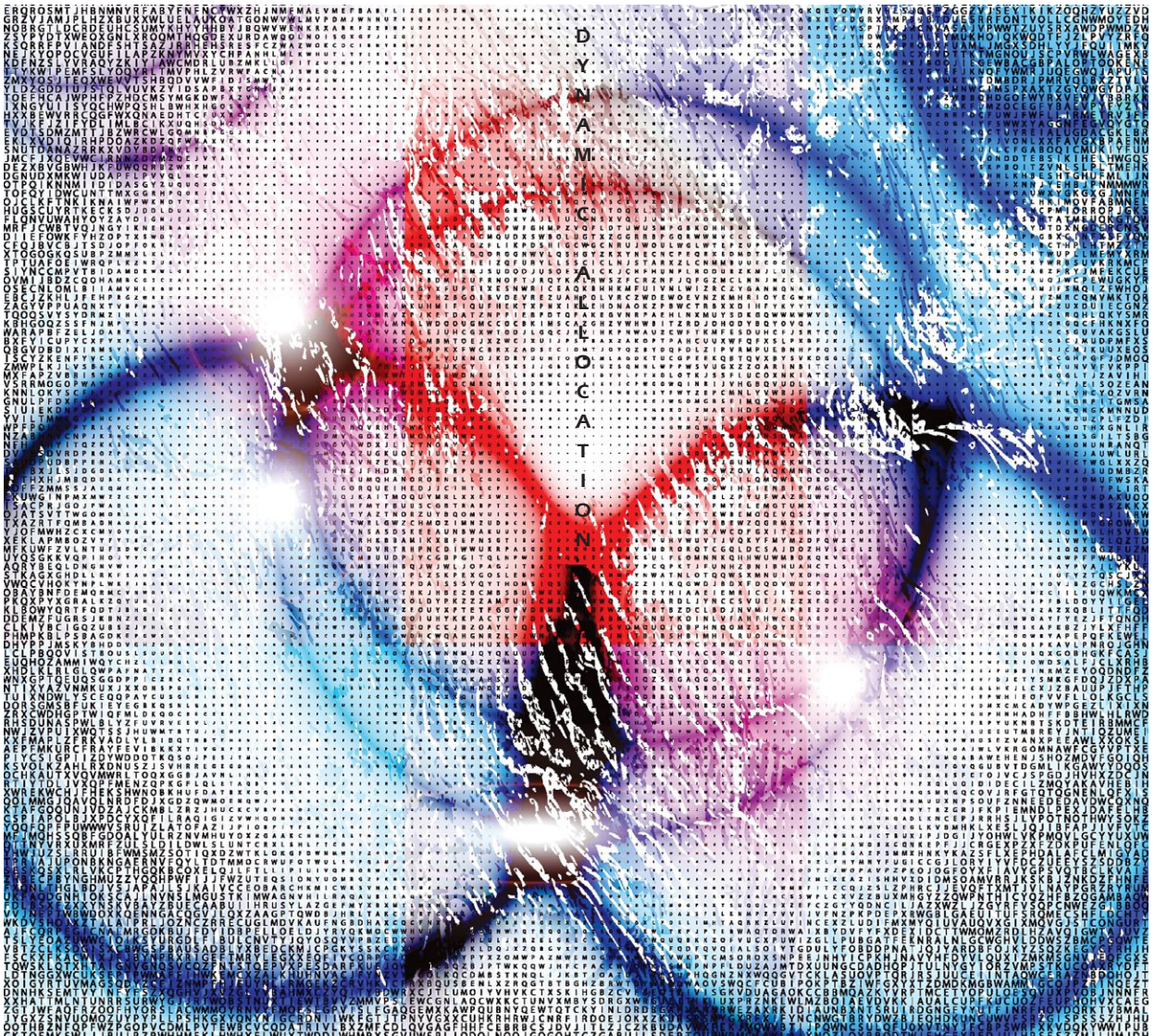
YEAR SUB

\$6.99

SINGLE COPY

Read It Today!





Improving MemGuard Support for UMA on FreeBSD

1. Background

There are two main ways to dynamically allocate memory in the FreeBSD kernel: malloc type and UMA zone. Malloc type uses `malloc()`, `realloc()`, and `free()` function to allocate and free memory. These functions are similar to their counterpart functions in the user space. UMA zone is more flexible than malloc type. Actually, any malloc type is implemented in one of the UMA zones.

However, there are three common bugs when

we use dynamic allocation memory in the kernel. These bugs are usually hard to find during testing and then turn into security bugs in the release code [1,2,3]:

- **read-before-initialization**: some fields in the buffer are used directly after allocation without initialization to correct values. If users request the zero-filled memory (`malloc()/uma_zalloc()` with `M_ZERO` flag), using the pointer in the buffer causes null pointer dereference. On the other hand, users do not specify the `M_ZERO` flag, but assume the buffer is zero-filled. This can cause logical errors and subsequent bugs.

- **use-after-free:** either read or write the memory that is already freed. This kind of bug is a race condition. For example, user A frees a memory buffer. Then the memory is allocated and written by another user B. If user A reads the memory buffer again, this may cause a bug. If user A also writes the memory buffer, then user B will use the corrupt value in the buffer.

- **memory-overflow:** either read or write the memory outside of the current memory buffer. The most common one is memory overflow, which accesses the memory after the memory buffer. The less common one is memory underflow, which accesses memory before the memory buffer.

These bugs usually cause strange behaviors or kernel panics. There are three debug options in the FreeBSD kernel to detect these errors:

1.1 Invariants

INVARIANTS is a set of detection that developers add to detect error conditions. In 2002, jeff added some checks into functions `malloc()` and `free()`. When a malloc-type memory buffer is first allocated from virtual memory, the memory is initialized with `0xdead0de`. After calling `malloc()`, if users forget to initialize the allocated memory or assume the memory is all zero, this may cause strange behaviors or panic. For example, some variables have the value of `0xdead0de` or part of it. What's worse is kernel panic when accessing the address `0xdead0de`. In this way, the users know there is a use-before-initialization bug in the use of the variable.

When a memory buffer is freed, `free()` saves the malloc type pointer at the end of the memory and overwrites the rest of the memory with `0xdead0de`. When the memory is allocated again, `malloc()` checks that the memory is still `0xdead0de`. If not, the last malloc type is reported for tracking the bug. In this way, it can detect a write-after-free bug.

1.2 RedZone

In 2006, pjd added the facility to detect memory-overflow. In `malloc()`, the stack trace is saved in the buffer. There are also two additional 16 bytes buffer of value `0x42` allocated before and after the buffer. In `free()`, the two additional 16 bytes are checked. If the value is

not still `0x42`, there is buffer overflow or underflow. The saved stack trace and current backtrace are printed for debugging.

RedZone has a better memory-overflow detection rate than MemGuard, but RedZone can only protect malloc-type memory. RedZone is not compatible with MemGuard. If the memory is allocated by MemGuard, RedZone cannot protect the memory.

1.3 MemGuard

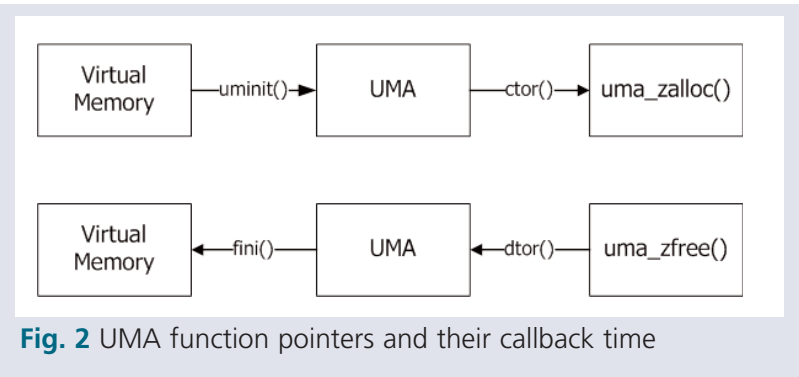
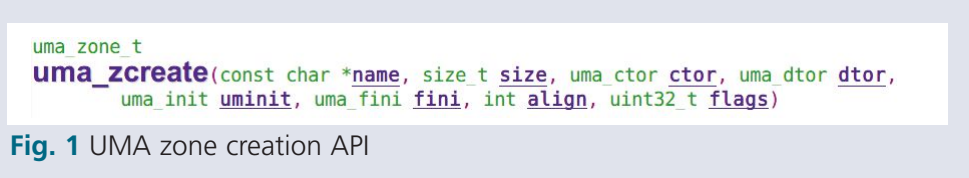
In 2005, bmilekic added the MemGuard to detect a write-after-free bug. A set of kernel memory is reserved for use in MemGuard. When the memory is freed, `free()` sets the memory page into read-only via `vm_map_protect()`. The freed memory is returned to the reserved set of MemGuard memory. Further writes to the freed memory would result in panic. When the freed memory is once again allocated, `malloc()` sets the memory page to read-write.

In 2010, mdf enhanced MemGuard to detect the read-after-free bug. When the memory is allocated, `malloc()` uses `vm_map_findspace()` to find the kernel virtual address in the vm map of MemGuard, and uses `kmem_back()` to allocate the physical address. When the memory is freed, `free()` uses `vm_map_delete()` to free the physical address. This mechanism greatly reduces the physical memory usage compared with the previous version. The mechanism can detect not only a write-after-free bug, but also a read-after-free bug. If there is further access to the freed memory, this would result in page-fault violation.

MemGuard can also detect memory-overflow when using the `MG_GUARD_AROUND` option. When the option is on, MemGuard adds two guard pages for each memory allocation. One guard page is placed before the allocation memory and another one is placed after the allocation memory. The guard pages allocate kernel virtual memory but not physical memory. If there is memory-overflow and the guard pages are accessed, this causes page-fault violation. Because memory overflow is more common than underflow when the allocation size is less than one page, MemGuard places the buffer near the end of a page. This increases the chance to detect memory overflow. Although RedZone has better memory-overflow detection rate, it can only detect memory-overflow write (MemGuard can detect memory-overflow read).

Tools	Allocation type	Read-before-init	Use-after-free	Memory-overrun
INVARIANTS	Malloc/UMA	Maybe	Write	
RedZone	Malloc			Write
MemGuard	Malloc/UMA	Maybe	Read/Write	Read/write

Table 1: Three kernel debug options and their protection types. In 2011, glebius added support for UMA (Universal Memory allocator). This gives MemGuard more coverage than RedZone (which only protects malloc type). Table 1 compares these kernel debug options and their protection types.



2. UMA

UMA is also called a zone or slab allocator in other OS. A zone can be created for a struct/object for efficient reuse. As shown in Figure 1, the `uma_zcreate()` API can create a new zone. The creation API can specify the zone name, size, four function pointers, align, and flags. The four function pointers are used for customization callback. Malloc types are also allocated from malloc-type zones in UMA but without using these function pointers (unless defined INVARIANTS). Figure 2 shows the four function pointers and their callback times.

- **initializer:** when UMA allocates objects from VM (virtual memory), the `init()` is called. This is usually used to initialize lock.
- **constructor:** when `uma_zalloc()` allocates objects from UMA, the `ctor()` is called. This is usually used to initialize other values in the object. In normal conditions, for each memory in UMA, the `init()` is only called once, but the `ctor()` is called each time the memory is reused in UMA.
- **destructor:** when `uma_zfree()` frees the objects and returns them to UMA, the `dtor()` is called. Afterwards the memory is returned back to UMA; it is not returned to the VM immediately. The free memory in UMA is

ready for efficient reuse.

- **finalizer:** when VM has low memory, it drains UMA to return free memory and `fini()` is called.

In normal memory load conditions, each memory life cycle includes one `init()` call and many `ctor()/dtor()` calls. When there is memory pressure, the `fini()` is called and the memory life cycle needs to restart again.

3. MemGuard enhancement for UMA

Most UMA zones can be protected by current implementation of MemGuard, but there are some issues to be addressed.

3.1 zones with init() and fini()

When MemGuard starts to support UMA, the four function pointers in the zone need to be called. This is to emulate the UMA behavior, but there is a difference. For each memory allocation, `init()` and `ctor()` are called; for each memory free, `dtor()` and `fini()` are called. This is used for detecting more errors rather than for memory efficiency.

In current MemGuard, `uma_zalloc()` calls `zone->uz_init()` and `zone->uz_ctor()` after allocating memory; `uma_zfree()` calls `zone->uz_dtor()` and `zone->uz_fini()` when freeing memory. However, the `zone->init()` and `zone->fini()` are not the real initializer and finalizer of the zone. For zones that use `init()` and `fini()`, this may cause bugs when these zones are protected by MemGuard. FreeBSD zones are backed by the keg system. So we should use `keg->uk_init()` and `keg->uk_fini()` instead.

3.2 zones with UMA_ZONE_PCPU flag

These zones are allocated for use in per-cpu counters. For example, zone "64 pcpu" is created with size 8 bytes. In addition to the counter, there is also a struct pcpu for each CPU. When the zone is protected by MemGuard, only the counter itself is allocated. So there would be a false alarm of memory overflow. We should add the size with `sizeof(pcpu) * mp_ncpu`.

3.3 realloc() with M_WAIT_OK

In the current MemGuard, when `realloc()` calls on the memory buffer allocated from MemGuard, it would call `memguard_alloc()` to allocate another memory. But if the `memguard_alloc()` fails, `realloc()` returns NULL even with `M_WAIT_OK` flag. This could result in panic for the

`realloc()` caller.

We change the behavior. If `realloc()` with `M_WAIT_OK` flag calls `memguard_alloc()` but fails, it would call normal `malloc()` instead.

3.4 lock already init

The `UMA_init()` function usually initializes the lock in the allocated object. But `lock_init()` would assert fail with the message "lock already initialized" when the memory appears to like an initialized lock. The assertion fails easily when MemGuard is enabled.

There are two solutions to this problem:

- `bzero()` the allocated memory before `lock_init()`
- use the `LO_NEW` flag in `lock_init()`. This can be done with `_NEW` flag when lock init. `mtx_init()` uses `MTX_NEW` flag. `rm_init()` uses `RM_NEW` flag. `rw_init()` uses `RW_NEW` flag. `sx_init()` uses `SX_NEW` flags.

ServerU

Rack-mount networking server

Designed for BSD and Linux Systems
Up to **5.5Gbit/s** routing power!

Made for  FreeBSD



PERFECT FOR

- BGP & OSPF routing
- Firewall & UTM Security Appliances
- Intrusion Detection & WAF
- CDN & Web Cache / Proxy
- E-mail Server & SMTP Filtering
- Anti-DDoS and clean pipe filtering

KEY FEATURES

- 6 NICs w/ Intel igb(4) driver w/ bypass
- Hand-picked server chipsets
- Netmap Ready (FreeBSD & pfSense)
- Up to 14 Gigabit expansion ports
- Up to 4x10GbE SFP+ expansion



Designed. Certified. Supported

1 Gbit/s Copper	Ports	Chipset
L800-G808-1	8x Gbe RJ-45 ports	8x Intel i210 AT; PEX8618
L800-G808-2	8x Gbe RJ-45 ports	8x Intel i210 AT; PEX8618
L800-G428-1	4x Gbe RJ-45 ports	1x Intel i350 AM4
L800-G428-2	4x Gbe RJ-45 ports	1x Intel i350 AM4
1 Gbit/s SFP (Fiber)	Ports	Chipset
L800-S406-1	4x Gbe SFP ports	i350-AM4
10GbE Copper	Ports	Chipset
L800-T202-1	2x 10Gb RJ-45 ports	Intel X540
L800-T203-1	2x 10Gb RJ-45 ports	Intel X540
10GbE SFP+ (Fiber)	Ports	Chipset
L800-X204-1	2x 10Gb SFP+	Intel 82599ES
L800-X205-1	2x 10Gb SFP+	Intel 82599ES
L800-X405-1	4x 10Gb SFP+	Intel 82599ES; PEX8724

3.5 Unsupported zones

Even with the above modifications, there are still some zones that cannot be protected by MemGuard.

UMA_ZONE_REFCOUNT: these zones use a field in the union plinks of struct `vm_page` to store the reference count. MemGuard uses the same field to store requested size and allocated size. The feature in these zones conflicts with the feature of MemGuard.

UMA_ZFLAG_BUCKET and **UMA_ZONE_VM:** these zones are used with bucket and VM. When these zones are protected by MemGuard, this results in a recursive lock problem.

```
377 void
378 pipe_dtor(struct pipe *dpipe)
379 {
380     ino_t ino;
381
382     ino = dpipe->pipe_ino;
383     funsetown(&dpipe->pipe_sigio);
384     pipeclose(dpipe);
385     if (dpipe->pipe_state & PIPE_NAMED) {
386         dpipe = dpipe->pipe_peer;
387         funsetown(&dpipe->pipe_sigio);
388         pipeclose(dpipe);
389     }
```

Fig. 3 use-after-free bug in `pipe_dtor()`.

4. Bugs Found

MemGuard is a dynamic detection tool.

It needs test cases to trigger bugs. Kernel compiling is a good test for MemGuard. Test cases in `tools/regression/` are very good too. With these test cases, four new bugs are found in the FreeBSD kernel.:

```
521     crp->crp_etype = 0;
522     err = crypto_dispatch(crp);
523     if (err != 0 && error == 0)
524         error = err;
525 }
526 if (bp->bio_error == 0)
527     bp->bio_error = error;
```

Fig. 4 use-after-free bug in `geli`.

4.1 use-after-free in `pipe_dtor()`

As shown in Figure 3, `pipe_dtor()` calls `pipeclose(dpipe)` to free the `dpipe`. But `dpipe->pipe_state` is used in the next line. This results in a use-after-free bug [4]. This bug existed for three years before it was found by MemGuard. This proves that this kind of bug is hard to find and MemGuard can do the job.

4.2 use-after-free in `g_eli_auth_run()` and `g_eli_crypto_run`

Two functions, `g_eli_auth_run()` and `g_eli_crypto_run()`, in the `geli` module have use-after-free bugs [5]. These functions call `crypto_dispatch()` to send crypto requests. There is a race condition here. After the last child bio is served, the bp is freed in `g_vfs_done()`.

As shown in Figure 4, functions `g_eli_auth_run()` and `g_eli_crypto_run()` use the freed bp. Setting error value to the freed bp may cause memory corruption.

4.3 use-before-init in `seltdinit()`

As shown in Figure 5, `thread_init()` is the init function of `THREAD` zone. When `thread_alloc()` allocates a struct `thread` from the zone, the field `td_sel` is not initialized. Later in `seltdinit()`, if `td_sel` is not NULL, this field will not allocate memory and causes panic later on. The `thread->td_sel` field is not initialized, but is used in `seltdinit()`. This is a use-before-init bug [6].

4.4 use-before-init in `dmu_buf_init_user()`

When the `sa_cache` zone first allocates memory from VM, `sa_cache_constructor()` does not initialize the `dbu_evict_func` that contains garbage. This will trigger the assertion. The `handle->db_buf->dbuf_evict_func` is not initialized. But the field is used in

```

198 static int
199 thread_init(void *mem, int size, int flags)
200 {
201     struct thread *td;
202
203     td = (struct thread *)mem;
204
205     td->td_sleepqueue = sleepq_alloc();
206     td->td_turnstile = turnstile_alloc();
207     td->td_rlqe = NULL;
208     EVENTHANDLER_INVOKE(thread_init, td);
209     td->td_sched = (struct td_sched *)&td[11];
210     umtx_thread_init(td);
211     td->td_kstack = 0;
212     return (0);
213 }
214
1823 static void
1824 seltdinit(struct thread *td)
1825 {
1826     struct seltd *stp;
1827
1828     if ((stp = td->td_sel) != NULL)
1829         goto out;
1830     td->td_sel = stp = malloc(sizeof(*stp), M_SELECT, M_WAITOK|M_ZERO);
1831     mtx_init(&stp->st_mtx, "selck", NULL, MTX_DEF);
1832     cv_init(&stp->st_wait, "select");
1833 out:
1834     stp->st_flags = 0;
1835     STAILQ_INIT(&stp->st_selq);
1836 }

```

Fig. 5 use-before-init in seltdinit()

```

209 static int
210 sa_cache_constructor(void *buf, void *unused, int kmflag)
211 {
212     sa_handle_t *hdl = buf;
213
214     mutex_init(&hdl->sa_lock, NULL, MUTEX_DEFAULT, NULL);
215     return (0);
216 }
217
1387 handle = kmem_cache_alloc(sa_cache, KM_SLEEP);
1388 handle->sa_userp = userp;
1389 handle->sa_bonus = db;
1390 handle->sa_os = os;
1391 handle->sa_spill = NULL;
1392 handle->sa_bonus_tab = NULL;
1393 handle->sa_spill_tab = NULL;
1394
1395 error = sa_build_index(handle, SA_BONUS);
1396
1397 if (hdl_type == SA_HDL_SHARED) {
1398     dmu_buf_init_user(&handle->sa_dbu, sa_evict, NULL);
1399 }
1400
561 inline void
562 dmu_buf_init_user(dmu_buf_user_t *dbu, dmu_buf_evict_func_t *evict_func,
563                 dmu_buf_t **clear_on_evict_dbufp)
564 {
565     ASSERT(dbu->dbu_evict_func == NULL);

```

Fig. 6 use-before-init in dmu_buf_init_user().

dmu_buf_init_user(). This results in a use-before-init bug [7]. Only after the memory is freed, the sa_cache_destructor() sets dbu_evict_func to NULL and returns it to the zone for next use.

5. Conclusion

MemGuard is effective for dynamic detection of memory error. In this work, some issues in the current MemGuard implementation are addressed. Four new bugs are found when compiling the kernel and running regression test cases. This makes MemGuard a very good tool for use when developing and testing.

We need more test cases to expand coverage. Currently, no memory-over-run bugs have been found via MemGuard. We will use fuzzer tools to inject garbage data to do more tests. •

LUKE CHANG-HSIEN TSAI received a masters of computer science from National Chiao Tung University, Taiwan. He has been a FreeBSD developer since 2012. At work he develops new features and improves ZFS performance. His spare time is used to submit patches to the FreeBSD Project, particularly in the areas of DTrace and security.

REFERENCES

- [1] FreeBSD Project, syncache/syncookies denial of service, <https://www.freebsd.org/security/advisories/FreeBSD-SA-02:20.syncache.asc>, April 2002
- [2] FreeBSD Project, kqueue pipe race conditions, <https://www.freebsd.org/security/advisories/FreeBSD-SA-09:13.pipe.asc>, October, 2009
- [3] FreeBSD Project, Buffer overflow in handling of UNIX socket addresses, <https://www.freebsd.org/security/advisories/FreeBSD-SA-11:05.unix.asc>, September 2011
- [4] FreeBSD Bugzilla, [patch] use-after-free bug in pipe_dtor(), https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=197246, February, 2015
- [5] FreeBSD Bugzilla, Bug 199705 - [patch] [geom] use-after-free bug in geli, https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=199705, April 2015
- [6] FreeBSD Bugzilla, Bug 199518 - [patch] use uninitialized field td_sel of struct thread, https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=199518, April 2015
- [7] FreeBSD Bugzilla, Bug 202358 - [patch] [zfs] fix possible assert fail in sa_handle_get_from_db(), https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=202358, August 2015

FreeBSD

Virtualization Options

Application and operating system containment is receiving unprecedented attention right now thanks to the hype surrounding technologies like Docker and all things “Cloud.” by Michael Dexter

BSD Unix has offered a compelling containment platform ever since Bill Joy “hardened” the `chroot(8)` command in 1981. From `jail(8)` in 2000 to `bhyve(8)` in 2014, FreeBSD has been quietly introducing elegant yet powerful containment and virtualization technologies that leverage FreeBSD’s natural containment environment, including its source-orientation, ports system, reduction tools like NanoBSD, `src.conf`, and `KERNCONF`, plus the unrivaled OpenZFS enterprise file system.

Containment, Meet Security

Just as `chroot(8)` proved in 1979, and Docker reminded us of in 2013, security is often an afterthought of containment. Neither of these technologies took security into consideration when conceived and have subsequently added various protections over time. While CSRG member Bill Joy simply prevented processes from escaping a `chroot(8)` environment, FreeBSD developer Poul-Henning Kamp took this work to the next level with the `jail(8)` utility in the year 2000, which enabled a stock FreeBSD operating system to be started within a “jail” container. The result was a lightweight means of running dozens, if not hundreds, of contained FreeBSD instances on a single FreeBSD host. I asked Poul-Henning at EuroBSDcon, in Copenhagen, if there was a theoretical limit to the number of jails one could run. He said that he “gave up at 64,000” when he used a script to continuously create minimal jails. With 15 years of proven production use, FreeBSD `jail(8)` has both saved and made money for operators around the world and has inspired containment systems on other operating systems such as Solaris and GNU/Linux. By institutionalizing containment, `jail(8)` set the stage for Capsicum, the FreeBSD capabilities framework, and is emerging as the preferred sandbox for every component of the FreeBSD operating system.

With all eyes turned to the Docker containment environment, it is no surprise that FreeBSD jail(8) is a natural fit for Docker's new pluggable container architecture along with the OpenZFS file system's snapshot and clone ability. Docker's model of transient and disposable application images demands institutionalized snapshotting and cloning at the file system level, and OpenZFS is just about the best option available. The irony is, however, that Docker solves the problem of consistent predictable application configuration that FreeBSD has traditionally never suffered from thanks to its source orientation and various software configuration tools. FreeBSD's "logic" with regard to software begins with the fact that the whole of FreeBSD in its self-hosting source form resides in /usr/src/ and is supplanted by third-party applications in the /usr/ports/ directory. The "bird's-eye view" alone that these two locations of software provide the operator is the first step in effectively managing software on FreeBSD. From there the operator can individually configure software "ports" and customize the entire operating system using the src.conf and KERNCONF kernel and userland configuration files. Where GNU/Linux distributions are additive collections of software from many sources, FreeBSD is a complete operating system from which parts are easily subtracted. To build the operating system without a compiler or name server is a simple matter of disabling those components in src.conf. Unnecessary hardware drivers can be excluded just as easily with a KERNCONF kernel configuration file. Together, the ability to granularly configure the FreeBSD userland, kernel and third party software makes FreeBSD a natural containment platform.

From Application Containment to Operating System Containment

Containing applications on FreeBSD assumes that they are comprised of FreeBSD native binaries. This assumption is challenged by the fact that FreeBSD's kernel-level Linux compatibility layer has seen many improvements in 2015, including the ability to execute 64-bit Linux binaries. This functionality has proven useful for Linux jail(8) environments and key to running native Linux Docker images. While Linux compatibility is suitable for many use cases, it does not meet all requirements to support foreign operating systems and their native software. This is where virtualization comes into play, and FreeBSD has made significant progress in this area in the last

two years. To begin with, FreeBSD has been well supported as a guest operating system on Amazon's Xen-based EC2 platform, thanks to the work of Tarsnap's Colin Percival. FreeBSD and the Xen hypervisor, however, saw their biggest breakthrough earlier this year when FreeBSD gained the ability to serve as a Xen Dom0 control domain host. This development allows FreeBSD to draw from the wealth of the mature Xen community and support most, if not all, DomU virtual machine domains. Unlike most Xen implementations, FreeBSD is forward-looking by only supporting hardware-assisted HVM domains. This move shifts the majority of the burden of executing guest virtual machines to the virtualization features found in modern CPUs. The result is bare-metal virtual machine performance with the added benefit of the flexible storage infrastructure that OpenZFS provides.

The Hardware-assisted Future

If we consider a hardware-assisted FreeBSD Xen Dom0 host with an OpenZFS-based storage architecture a glimpse of the future of virtualization, that future has in fact arrived on FreeBSD with the bhyve hypervisor. bhyve, the BSD Hypervisor, officially debuted in the FreeBSD 10-RELEASE and has been gradually adding features such as support for AMD processors and 32-bit virtual machines. Unlike Xen and Linux KVM, bhyve foregoes BIOS, video and network device emulation in favor of userspace bootloaders the virtualization-native VirtIO driver model. Like Xen on FreeBSD, bhyve relies exclusively on the modern virtualization extensions found in recent CPUs, resulting in bare-metal VM performance. Currently, bhyve supports FreeBSD, OpenBSD, NetBSD and GNU/Linux virtual machines and as of the recent EuroBSDcon conference, Microsoft Windows and Joyent SmartOS.

FreeBSD has long played a key yet humble role in Unix application and operating system containment and is poised to provide compelling virtualization options for organizations of all sizes. ●

MICHAEL DEXTER has used BSD Unix systems since January of 1991 and provides BSD and ZFS support at Gainframe. He has supported BSD Unix with download mirrors, projects, events, and organizations for over a decade. He lives with his wife, daughter, and son in Portland, Oregon.

conference **REPORT**TM

by Dr. Robert N. M. Watson

BSDCam 2015 (<https://wiki.FreeBSD.org/201508DevSummit>)

BSDCam 2015, the annual FreeBSD Developer Summit in Cambridge, England, took place at the University of Cambridge Computer Laboratory, August 17–19, 2015. Hosted and organized by the University of Cambridge, BSDCam was also supported by grants from the FreeBSD Foundation and Xinuos.



The developer summit was attended by over 50 FreeBSD developers, local PhD students and researchers, and guests from tech companies using (or otherwise interested in) FreeBSD.

The Computer Laboratory uses FreeBSD extensively in its research, especially into security, networking, operating-system, program analysis, and computer-architecture research—all topics that were raised throughout the summit. Cambridge is also home to ARM Ltd, which develops the ARM architecture; the company was well represented at BSDCam, leading sessions on power management and embedded architectures, as well as contributing to sessions on ABIs, toolchain, and processor-based counters and tracing. Other organizations represented at the event included BAE Systems, EMC, the FreeBSD Foundation, Netflix, Neville-Neil Consulting, Nuxi, ScaleEngine, SRI International, Tarsnap, the University of Texas at Austin, Wheel Systems, and Xinuos.

The Computer Laboratory is well suited to host developer summits, with a range of well-equipped meeting rooms capable of hosting the entire group, as well as accommodating smaller breakout sessions. Run “un-conference-style,” BSDCam began with a brainstorming session on topics of interest, providing a schedule with one or two concurrent sessions throughout most of the event. Developers continued chatting well into the evening, whether at the pub or, on the second evening, a formal conference dinner at

Murray Edwards College, sponsored by Xinuos. Daytime sessions ran one to two hours, with topics including:

- Future of the storage subsystem
- Network-stack performance
- The package system
- Toolchain/LLVM
- ABI emulation
- Software tracing and debugging
- Testing and QA
- Power management
- Capsicum and CloudABI
- Security, crypto, and the RNG
- Teaching with FreeBSD
- FreeBSD in the cloud
- FreeBSD/RISC-V
- FreeBSD/ARMv8

Detailed write-ups of sessions, as well as slides from many of the talks, can be found on the BSDCam 2015 wiki page (<https://wiki.FreeBSD.org/201508DevSummit>), and summaries of a selection of sessions follow.

BSDCam would not have been possible without the contributions of Jon Anderson (Memorial University Newfoundland), Pieter Brooks (University of Cambridge), David Chisnall (University of Cambridge), Brooks Davis (SRI International), George Neville-Neil (Neville-Neil Consulting), Robert Watson (University of Cambridge), and Bjoern Zeeb (University of Cambridge).

Tracing and Networking

Network-stack evolution and performance tracing/profiling have been recurring themes at BSDCam for the last few years, given both local research interest in Cambridge, and the interest of individuals and companies attending the event. George Neville-Neil and Robert Watson chaired the tracing session, with Al Grant (ARM) kicking it off with a tour of ARM's Streamline and Coresight IP blocks. These tools provide hardware-assisted tracing and performance visualization for ARM-based systems, capturing architectural and micro-architectural events, as well as software-generated events, and providing a conducive UI for performance debugging. ARM and Intel have been attempting to rationalize their approaches in order to provide greater tool portability across event sources. The more general discussion that followed touched on topics such as DTrace optimization, distributed DTrace, and introducing machine-readable output for the DTrace command-line tool, which has historically been focused on user-facing interaction.

It was clear that substantial refactoring is overdue for the DTrace library and tool in order to allow output to have an object model—but the libxo library, normally used for machine-readable output in FreeBSD, will likely require work to better support streaming data sources, as well as schemas. Another topic discussed was how to provide basic integration of CPU performance counters into the DTrace framework—but this is complicated by conceptual mismatches between HWPMC, FreeBSD's current PMC framework, and DTrace. There was also concern that DTrace might introduce additional overhead, increasing the probe effect when using performance counters.

George Neville-Neil chaired the network-stack session, which touched on a broad range of topics. Particular focus was placed on continuing to improve network-stack scalability, with a detailed discussion of current known multicore scalability bottlenecks, especially in routing decisions and lower layers of the stack (the best routing lookup is one that is not done at all!). There was also a substantial discussion of TCP feature improvements, and how to better test the TCP stack for regressions during feature enhancement, as subtle problems have arisen in the past that can be very difficult to diagnose and debug in the field. Past work at Cambridge (<http://www.cl.cam.ac.uk/~pes20/Netsem/>) on modeling the TCP state machine along with testing based on tracing may see new use today given improvements in tracing technology—but will need updating. Another topic of discussion was how to expose link-layer flow-control events to TCP in order to handle data-center networks (which can often be lossless) better.

Similarly, there is a desire to support link-layer flow control better in userspace applications and network stacks—possibly via kqueue improvements.

Toolchain/LLVM, and ABIs

Ed Maste chaired the session on toolchain and LLVM, which reviewed the current state of the toolchain and ongoing development efforts in this space. Quite a bit of time was spent discussing the extent of LLVM integration and areas where further improvement is required (e.g., for 64-bit MIPS), and how LLVM is now facilitating research (e.g., at SRI, Cambridge, Memorial, and BAE)—including via patches to generate LLVM IR output as part of the FreeBSD build process. The status of the BSD ELF toolchain was also discussed—both the enormous growth in maturity, and some continuing gaps, such as complete support for objcopy and objdump's features. The LLVM linker world seems to be starting to settle on LLD, and the FreeBSD community has started to contribute to this more substantially. The LLDB port continues to improve, with structural changes being made upstream in LLDB to support multiple operating systems better. There was also discussion of whether to continue to use CTF for DTrace, or attempt to use DWARF directly—it seems that the status quo will continue here, but BSD-licensed CTF generation tools would be useful to have. LLVM support in the ports collection is being driven by the arm64 port, which uses only LLVM, not gcc—but with consequences for programming languages such as Fortran.

Allan Jude and Johannes Meixner led the ABI session, which focused on current work on the Linux emulator, as well as other work on ABI compatibility. The Linuxulator developers are continuing to pursue support for Centos 6.7 (both 32-bit and 64-bit). (Lack of) support for inotify seems to be a continuing blocker for ports such as Dropbox, and there are other functional omissions affecting support for tools such as Google Hangouts, but also the Xilinx and Altera FPGA toolchains. Ed Schouten presented his work on CloudABI, a system-call interface designed

Over 40 developers attended the developer summit at the University of Cambridge.



conference **REPORT**

for Capsicum, and now present in FreeBSD. Brooks Davis presented his work on CheriABI, a system-call ABI intended to support userspace fat pointers found in the ChERI processor.

Capsicum, CloudABI, and Security

Robert Watson and Ed Schouten chaired the session on Capsicum, which was also joined by Jon Anderson (Memorial) and Ben Laurie (Google) via teleconference. Topics of discussion included the Linux Capsicum effort at Google, convergence of concepts such as Apple's launchd and the runtime linker, the need for better (and more tutorial-like) documentation for developers, Alex Richardson's work on KDE support for Capsicum, and the lack of a suitable RPC framework. Jon Anderson described the SOAAP project, which uses LLVM

static analysis to model sandboxing in applications, helping to chart potential improvements but also to find bugs—work to be published at ACM CCS 2015. There was also discussion of capability “reification”—the ability to create persisting entries in the file system derived from capabilities, and an issue with linkat() requiring two permissions (from, to), similar to rename(), rather than the single permission used today, to avoid bypass of directory capability permissions.

The security session, chaired by Robert Watson, focused primarily on multicore scalability improvements to FreeBSD's new Fortuna-based random number generator. There has been continuing concern about the performance overhead of entropy gathering, but also how to avoid multicore communication at various points in the random-number-generation pipeline. Mark Murray led the discussion about a number of models and limitations, with the eventual conclusion that per-socket (or perhaps even per-core) Fortuna instances would be required. This does require addressing some hard problems, such as how to address asymmetric distribution of entropy sources across sockets or cores—which might require a rebalancing mechanism. Regardless, the goal is to continue to reduce the overhead of the kernel random number generator, especially in entropy harvesting.

Developers gathered from around Europe (and the world) to discuss topics ranging from network-stacks to ARMv8.



FreeBSD Journal

The **NEW** publication for FreeBSD buffs, is **OLD** enough to have

10 BACK ISSUES!



read all you missed!

Order as desktop editions at www.freebsd.foundation.org
or purchase at your favorite app store.

ARMv8 and RISC-V

Two sessions addressed two new Instruction-Set Architectures (ISAs) for which FreeBSD support is being developed: ARM's 64-bit ARMv8 ISA, and the University of California at Berkeley's open-source RISC-V ISA, led respectively by Andrew Turner (FreeBSD Foundation—sponsored by ARM and Cavium) and Ruslan Bukin (University of Cambridge).

Andrew described his ongoing work on the FreeBSD/ARMv8 port being sponsored by the FreeBSD Foundation, ARM, and Cavium, which is also being done in collaboration with Semihalf. FreeBSD now boots and runs well on a number of 64-bit ARM devices including evaluation boards from ARM and Cavium. Work continues to mature the port in terms of improving stability and functionality. Sponsored by ARM, Ruslan Bukin has recently completed initial support for Hardware Performance Counters (HWPMC) and DTrace for ARMv8, based on his earlier work for ARMv7. FreeBSD is also usable on multicore Cavium Thunder boards, thanks to support for Semihalf from Cavium.

Ruslan Bukin and Ed Maste led the session on the budding RISC-V port of FreeBSD. RISC-V is a new "open-source ISA" created by the the RISC-V team at the University of California at Berkeley; RISC-V is

being used for a number of processors there, and is also the basis of the LowRISC open-source SoC being developed at Cambridge. It will be used in other processor research and embedded systems in the future. Ruslan now has FreeBSD/RISC-V printing boot messages in Berkeley's Spike simulator, and is starting work on context switching and virtual-memory support. He has encountered some difficulties as the RISC-V instruction set is still maturing—for example, QEMU implements an older (and incompatible) version of the privileged ISA, and there have been issues with toolchain. However, progress is being made rapidly. ●

DR. ROBERT N. M. WATSON is a University Lecturer in Systems, Security, and Architecture at the University of Cambridge Computer Laboratory; FreeBSD developer and core team member; and member of the FreeBSD Foundation board of directors. He leads a number of cross-layer research projects spanning computer architecture, compilers, program analysis, program transformation, operating systems, networking, and security. Recent work includes the Capsicum security model, MAC Framework used for sandboxing in systems such as Junos and Apple iOS, and multithreading in the FreeBSD network stack. He is a coauthor of *The Design and Implementation of the FreeBSD Operating Systems (Second Edition)*.

ISILON The industry leader in Scale-Out Network Attached Storage (NAS)

Isilon is deeply invested in advancing FreeBSD performance and scalability. We are looking to hire and develop FreeBSD committers for kernel product development and to improve the Open Source Community.



We're Hiring!

With offices around the world, we likely have a job for you! Please visit our website at <http://www.emc.com/careers> or send direct inquiries to karl.augustine@isilon.com.



EMC²

ISILON

svn UPDATE

by Glen Barber

WITH THE FREEBSD 10.2-RELEASE out the door, albeit only recently, this edition of *svn update* highlights some of the exciting updates expected in upcoming FreeBSD releases. Enjoy!

IPsec in the GENERIC Kernel

<http://svnweb.freebsd.org/changeset/base/285142>

IPsec is a set of networking protocols that allow secure end-to-end communication between hosts on the Internet, encrypting each IP packet of communication. The implementation details of IPsec are covered in RFC4301.

After a significant number of updates to the underlying cryptography code and network stack, IPsec is now enabled by default in FreeBSD in all GENERIC kernel configurations.

Some of the updates prior to enabling IPsec by default include:

- Support for AES modes has been added for both software and hardware modes, allowing the use of hardware cryptography acceleration on systems where the AES-NI capability is supported by the CPU (r285336)—sponsored by Rubicon Communications (Netgate).
- Support for SKIPJACK was removed.
- Hardware acceleration support for the AES-GCM and AES-ICM cryptography was added.

Including IPsec in the GENERIC kernel allows consumers of third-party software that require the functionality, such as the security/ike port, to work by default without requiring the kernel to be rebuilt.

Initial NUMA Affinity and Policy Configuration

<http://svnweb.freebsd.org/changeset/base/285387>

Non-Uniform Memory Access, also known as NUMA, is a computer architecture design in which latency to specific memory or input/output devices depend on the processor to which the memory or device is attached. On multi-core systems, latency may be further increased if a processor needs to access a memory address or

device that is connected to another processor.

The first NUMA implementation on FreeBSD appeared in 9.0, but it was not configurable. As of r285387, the initial implementation of a configurable NUMA policy and affinity allocation for threads and processes was introduced—sponsored by Norse Corp. Inc. and Dell Inc.

The NUMA policies can be modified and queried with the `numactl(1)` utility, and the configuration managed by the `numa_getaffinity()` and `numa_setaffinity(2)` system calls.

Systems that do not define the `MAXMEMDOM` option in the kernel configuration file, or systems with `MAXMEMDOM=1`, are not affected by this change, but the system can be configured to take advantage of NUMA policy and affinity tuning by setting `MAXMEMDOM` to a value greater than '1' in the kernel configuration file. (See the `numa(4)` manual page for additional information.)

jail(8) Usage Updates

<http://svnweb.freebsd.org/changeset/base/286064>

The `jail(8)` utility is used to create, modify, or remove existing jails on the running system. The jail (or “prison”) is a `chroot(8)` environment with its own IP address space, user accounts, and processes, which can be used for process containment or separation from both the running system and other jails.

The `jexec(8)` userland utility is used to interact with the running jail by specifying a command to run within the target jail. Historically, a command to run within the jail was specifically required, even if the target command was a target shell within the jail. As of r286064, `jexec(8)` will run a shell in the target jail if no command arguments are specified. In addition, a new flag was added, `-l`, which discards all environment variables from the login class, except `HOME`, `SHELL`, `TERM`, and `USER`.

Toolchain Components Updated

<http://svnweb.freebsd.org/changeset/base/288943>

The major contributed toolchain components have all been updated, and are now all in line with the upstream versions. The affected compo-

nents include clang, llvm, lldb, compiler_rt, and libc++, all of which are now updated to the upstream version 3.7.0.

The release notes for these components can be found at:

- <http://llvm.org/releases/3.7.0/docs/ReleaseNotes.html>
- <http://llvm.org/releases/3.7.0/tools/clang/docs/ReleaseNotes.html>

bhyve(4) Enhancements

<http://svnweb.freebsd.org/changeset/base/288522>

bhyve(4) is the BSD-licensed hypervisor natively available in FreeBSD since 10.0-RELEASE.

The bhyve hypervisor recently had a number of interesting improvements, most notably in enhanced UEFI support. What is really intriguing about the update, however, is that it allows preliminary support of running Windows within bhyve.

Installing Windows Server is easier at the moment, as it allows a headless, serial console installation whereas the desktop versions of Windows do not.

Please see the original announcement email for details, and links to the How-To documentation at: <https://lists.freebsd.org/pipermail/freebsd-virtualization/2015-October/003832.html>.

EOF

Thank you, dear reader, for supporting the FreeBSD community, the *FreeBSD Journal*, and, of course, The FreeBSD Foundation.

Don't forget that development ISO and preinstalled virtual machine images (in VHD, VMDK, QCOW2, and RAW formats) of the FreeBSD-CURRENT and FreeBSD-STABLE branches—found on the FTP mirrors—are built weekly: <ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/>. As always, development snapshots are not intended for production use; however, we do encourage regular testing so we can make the next FreeBSD releases as great as you expect them to be. ●

As a hobbyist, GLEN BARBER became heavily involved with the FreeBSD Project around 2007. Since then, he has been involved with various functions, and his latest roles have allowed him to focus on systems administration and release engineering in the Project. Glen lives in Pennsylvania, USA.



atlantic.net

FAST SSD CLOUD HOSTING

Up in 30 Seconds

24/7 Support

Per Second Billing



One-Click Apps



nodejs

cPanel

LAMP

LEMP

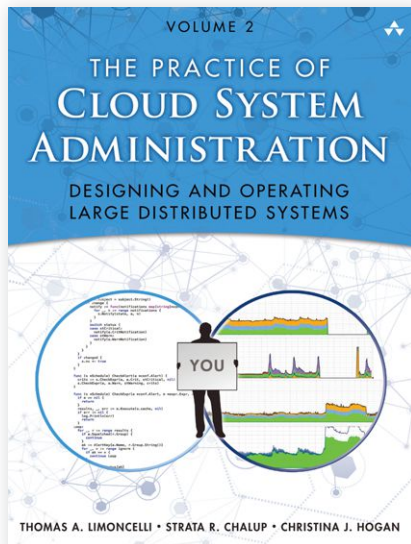
Starting at

\$4.97

per month

BOOKreview

by Greg Lehey



The Practice of Cloud System Administration

by Thomas A. Limoncelli, Strata R. Chalup and Christina J. Hogan

PublisherAddison-Wesley Professional
Print List PriceUS \$54.99
eBook List Price.....US \$43.99
ISBN.....032194318X
Pages.....560

The *Practice of Cloud System Administration*, by Thomas A. Limoncelli, Strata R. Chalup, and Christina J. Hogan, carries the superscript Volume 2, certainly a reference to *The Practice of System and Network Administration* by the same authors, though it's not stated explicitly.

It took me a long time to relate to the book. This "cloud" buzzword irritated me somewhat. It wasn't until I finished reading it that the penny finally dropped. Three texts spread through the book tell you what it's really about.

First there's the subtitle: Designing and operating large distributed systems. Now that makes sense to me. Why the other title? Right at the beginning, the second text makes clear that the word "cloud" is a "marketing term that means different things to different people." The authors prefer the term "distributed computing." I agree. I get the impression that the authors and the publishers didn't see eye to eye on this matter.

The third text is right at the end, and it answers a further question: how do you learn about designing and administering large systems through a book? It takes a large number of people and complex interactions between them to get it right. And at the very end of the epilogue the authors write, "We hope this book gave you good things to think about...These are the early years...The rest is up to you."

Distributed systems aren't new. At Tandem Computers we were doing that 30 years ago. And in many ways, the FreeBSD Project, started in the early 1990s, is also a distributed environ-

ment. The computers are mainly in Silicon Valley, but the developers are spread around the world.

But that's not the kind of distributed system that this book deals with. It focuses very much on large web-based systems similar to Amazon, eBay, or Google. How can you learn to build a site like that from a book, no matter how good? You can't, of course. But this book presents a summary of current best practices to ensure that your site runs smoothly. It may seem a little disjointed in places, but that reflects life. The book looks at the big picture: how do you manage a large system when the people are not all in the same place, and probably not in the same time zone. It's divided into two parts: "Building It" and "Running It."

"Building It" starts with an overview of what distributed computing is. It goes into considerable detail about redundancy, load-balancing, and distributed data storage. It reintroduces the CAP principle from the first volume: "consistency, availability, partition tolerance: choose two."

Chapter 2, Designing for Operations, considers how to ensure that what you build will actually work. A certain Unix bias shows through here: GUI configuration tools? No thank you. We prefer to be able to keep the config files in a version control system. That bias was present in the first volume too, but not as clearly.

Chapter 3, Selecting a Service Platform, is interesting. It describes three aspects: What service are you providing (infrastructure, platform, or software)? Are you providing it on a real machine, a virtual machine, or a container (jail)?

Can you afford to put your private data where other people can gain access to it?

Chapter 4, Application Architectures, is concerned with how to distribute the application based mainly on web servers. One machine? Several machines? In one place? Across the globe? How do you balance the load? That's only part of the issue. Chapter 5, Design Patterns for Scaling, looks at the rest: How do you ensure that performance doesn't suffer as you grow? I found these chapters two of the most instructive chapters in the book.

Chapter 6, Design Patterns for Resiliency, addresses the fact that hardware and software fail. What are the main causes of failure? How do you ensure that the show goes on? One of the more interesting topics in this chapter is the recognition that, since hardware fails anyway and you have to provide for it, there's no point in buying expensive hardware: you can save money by using cheap hardware and letting it fail more often. An interesting example is where Google used to buy memory chips that had failed quality control and worked around the deficiencies. It also mentions human error as one of the biggest issues, but defers discussing how to deal with it.

The second part of the book ("Running It") starts with Chapter 7, Operations in a Distributed World, an overview of the operational structure with a strong emphasis on the team and well-defined processes. It introduces a number of new terms, but rather too many buzzwords for my liking.

This chapter delivers two important lessons. Firstly, doing things by hand doesn't scale. You can install software on a single machine by hand. If it's 1,000 machines, you need to automate as much as is possible (but no more). Then there's the service life cycle: how do you introduce new services with as little disruption as possible (answer: surreptitiously). How do you allocate your time?

Chapter 8 describes DevOps, which I see as a current method for ensuring communication between different groups of people. It makes some rather broad assumptions about "The Traditional Approach," which appears to be creating shrink-wrapped software, dumping it on the users and going away again. It suggests little feedback from users, via a support team designed to keep the users out of the developers' hair, and infrequent product updates. We all know that (I consider it part of the "Microsoft space"), but it's not the only way. It's not the way the FreeBSD Project does things, and in my experience it has been relatively uncommon. This is clearly another issue of perspective.

One thing that this chapter doesn't address sufficiently is the influence of time zones on a DevOps team. How do you arrange a phone conference

with a team spread out between, say, California, Sweden, and Australia? They're all about eight hours apart, and daylight savings in Australia is out of phase with the northern hemisphere. This has happened to me, but I don't know a solution beyond "Well, don't do that, then."

Chapters 9 and 10 are concerned with service delivery. They're less obviously related to distributed computing than the rest of the book; in many ways they remind me of the FreeBSD Project: develop and upgrade software, and keep the show running all the while. Again we have a number of terms. Some, like Release Candidate, are old friends, but others, like Cycle Time, take on a whole new meaning (how frequently a Flow completes). The methods described differ from the way the FreeBSD Project does it, of course, but they offer some interesting considerations.

Chapter 11, Upgrading Live Services, addresses the issue that you don't really want systems to go down because they're being upgraded. What if a new feature proves to be a flop or it introduces bugs that take the system down? The chapter goes through several methods to mitigate the damage. This topic was already discussed in Chapter 7, but here the emphasis is subtly different. It also addresses issues like live database schema updates and live code changes.

Getting machines (computers) to do things rather than doing them yourself is clearly a good idea, and it's essential for scalability. Again, Chapter 7 touched on this, but Chapter 12 deals with the topic and its dangers in detail. It also goes into areas that don't really sound like they have much to do with automation, such as version control systems and style guides.

Keeping a big project of any kind running requires clear documentation. Chapter 13, Design Documents, deals with the issues, with a particular emphasis on responsibility and sign-offs.

Things fail at random, but most people work a "day job." Chapter 14 addresses the issue of having people on call at all times and how to minimize the necessity for actually calling somebody out. Here time zones can be your friend: if the sun never sets on your organization, there's always somebody awake when a problem occurs. An allied issue is being prepared for disasters, which is addressed in Chapter 15. Again there's good insight into Google's training for disaster.

How can you tell how well things are running? Monitor it, of course. But the larger the system, the more data you can collect. Do you need the information immediately, or are you looking for statistics? Both have their place. Chapter 16 goes into what to monitor for what purpose, and Chapter 17 presents an architecture of a monitoring system. It

goes a long way beyond log files: depending on the information collected, you may want to get a human involved, maybe a second if the first doesn't react. If things don't go that wrong, how do you present it to an administrator so that he can understand it? Anybody who has looked through Postfix mail logs knows how easily you can lose the overview of what's really going on. This chapter presents a number of options.

One thing you want to monitor is whether your system is delivering adequate performance. Even if it is, is it going to stay that way? With rising traffic, performance can drop, and you need a way of planning for future capacity. That's the subject matter of Chapter 18. This chapter includes a fair amount of scary-looking mathematical equations, but non-mathematicians can learn a lot too. After all, we have computers to do the math. This chapter also goes into more detail about launching new services, this time from the viewpoint of scalability.

The final two chapters discuss issues that arguably could have come earlier in the book: what are we measuring, anyway? KPIs, or Key Performance Indicators. This might sound simple: fast, accurate response. But of course things aren't always that simple. What happens if my promised 15 GB of mail storage isn't available because

there's not enough disk space? Clearly that's a requirement. This chapter helps identify the important parameters rather than the obvious ones. Key performance is the goal. Chapter 20, Operational Excellence, discusses the attitudes and measurements necessary to attain it.

Finally, there are a number of appendices. Appendix A is a sort of planning checklist to ensure that nothing has been forgotten. Appendix B deals with the history of high availability, from a somewhat different perspective from my own. Appendix C discusses more scaling concepts, and Appendix D is a template for a design document.

So if you're designing the next Google or eBay, how should you treat this book? You'd be crazy to ignore it. You'd also be crazy to rely on it alone. Think of it as a presentation of current best practice. Refer to it in every iteration of your design process. Your product will be the better for it. •

GREG LEHEY is a retired kernel hacker. He is a committer and ex core team member of the FreeBSD Project, and was a committer to the NetBSD Project. He is the author of the Vinum volume manager. His books include *Porting UNIX Software* and *The Complete FreeBSD*.

Thank you!

The FreeBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.



The
FreeBSD
FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today!
freebsd.foundation.org/donate/

Iridium



Platinum



Gold



Silver



Please check out the full list of generous community investors at freebsd.foundation.org/donate/sponsors



FREE AND OPEN SOURCE SOFTWARE EXPO
AND TECHNOLOGY CONFERENCE

FOSSETCON

2015

Come out and participate in the Second Annual Fossetcon 2015
Florida's Only Free and Open Source Conference. With in
2 minutes of Downtown Disney and other great entertainment.

DAY 0

**FOOD, TRAINING,
WORKSHOPS AND CERTIFICATIONS**

DAY 1

**FOOD, KEYNOTES, EXPO HALL,
SPEAKER TRACKS**

DAY 2

**FOOD, KEYNOTES, EXPO HALL,
SPEAKER TRACKS**

BSD

Friendly

FREE FOOD,
TRAINING,
CERTIFICATIONS
AND GIVEAWAYS!!!

NOV 19 - NOV 21

Hilton Lake Buena Vista Orlando, FL

Fossetcon 2015: The Gateway To The Open Source Community

More info at

www.fossetcon.org

PORTSreport

by Frederic Culot

AS USUAL, the summer period (July–August) was relatively calm on the ports front. Although there was a decrease in activity of almost 25% when considering the number of commits applied to the tree, major improvements were made.

NEW PORTS COMMITTERS AND SAFEKEEPING

It is a delight to welcome two new/returning talents to the rank of ports committers.

- First, Jason Unovitch was granted a ports commit bit, with feld@, delphij@, and pgollucci@ as mentors. Jason (now known as junovitch@) has been so active, with more than a hundred commits, that he has already been freed from mentorship. Thanks for your hard work, Jason!
- Returning after a period of inactivity is Babak Farrokhi (farrokhi@). To ease his return, philip@, bap@, and mat@ will help with the mentoring process.

Some commit bits were also taken in for safekeeping, on committers' requests (xmj@, stefan@, and brix@).



PORTS TREE 21ST ANNIVERSARY

Our ports tree turned 21 on August 21! For those of you who missed it, a video (<http://youtu.be/LiFq5D-zmBs>) was created to celebrate the 20th anniversary of the ports tree last year.

Having completed his PhD in Computer Modeling applied to Physics, Frederic Culot has worked in the IT industry for the past 10 years. During his spare time he studies business and management and just completed an MBA. Frederic joined FreeBSD in 2010 as a ports committer, and since then has made around 2,000 commits, mentored six new committers, and now assumes the responsibilities of portmgr-secretary.

IMPORTANT PORTS UPDATES

Many exp-runs were performed by antoine@ and mat@ (15 in total) to verify the safety of major ports updates, including the following:

- Gnome updated to 3.16
- gettext updated to 0.19.5.1
- doxygen updated to 1.8.10
- libreoffice updated to 5.0.1
- ghc Haskell compiler updated to 7.10.2

As usual, please read the /usr/ports/UPDATING file carefully before updating your ports, as manual steps may be involved.

FIGURES

We would like to give a few figures so that people can get a better idea of the outstanding amount of work performed by our volunteers, even during the summer holidays period. During July and August, 4,212 commits were applied to the ports tree, 1,009 PR were closed, and 1,112 emails were received by portmgr@ (without counting the spam...). And all this was handled by an average of only 122 active ports developers!

Impressive figures, no doubt! Please note, however, that the ports tree is built and run by volunteers. So if you use ports or packages, please consider jumping in and helping! If you're an existing porter, consider stepping up to share your knowledge and mentor someone more junior with the ports tree internals. And if you already do these, many thanks to you!

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with  **BSD**
CERTIFICATION

Getting the most out of
BSD operating systems requires a
serious level of knowledge
and expertise ● ● ● ● ● ● ● ●

SHOW YOUR STUFF!

Your commitment and
dedication to achieving the
BSD ASSOCIATE CERTIFICATION
can bring you to the
attention of companies
that need your skills.

NEED AN EDGE?

● **BSD Certification can
make all the difference.**

Today's Internet is complex.
Companies need individuals with
proven skills to work on some of
the most advanced systems on
the Net. With BSD Certification

**YOU'LL HAVE
WHAT IT TAKES!**

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

this month

BY DRU LAVIGNE

FreeBSD has the reputation of being very well documented. The FreeBSD Project provides an extensive documentation set that includes the *FreeBSD Handbook*, the *Porter's Handbook*, the *Developer's Handbook*, the *Documentation Project Primer*, built-in and online man pages, and the website. Maintaining and translating this quantity of documentation presents its own challenges. This month, I had a chance to interview **WARREN BLOCK**. Warren is a longtime doc committer, a member of the FreeBSD DocEng team, and the author of *igor* (<http://www.wonkity.com/~wblock/igor/>).

Q Tell us a bit about yourself. How did you get started with FreeBSD and what is your involvement with the FreeBSD Project?



A In 1998, my subterranean volcano lair was getting tepid and the henchmen wanted to unionize, something about safety with the sharks and giant squid. It was all overblown; there was some kind of manufacturing defect with that trap door. Cletus was not the brightest henchman, either, and apparently did not really understand the lockout/tagout system. If you've ever priced giant squid chow, you know how that was also eating into the bottom line.

When looking for a lair operating system that was cheap but didn't have a licensing agreement that assumed I was a thief, a friend introduced me to FreeBSD version 4.0.

After that, it just went downhill. I installed FreeBSD on one system, then a couple more, then it was a few systems every month. This went on for years. FreeBSD was rich with application software, and I

used it all. I'm not too proud to say that sometimes I even took money for it.

After a long time, I got into bug reports, sometimes two or three in a single day, and often about problems with documentation. Doc committers are typical nerds, only perhaps more bookish, and working on those bug reports was cutting into the time they normally would spend posting snarky comments on mailing lists. At one point, Glen Barber and Benedict Reuschling proposed me for a doc commit bit. That is a rubber stamp saying, "It is possible you can commit directly to the docs without breaking them too badly, so go for it."

A bit more time passed, with no problems I caused that could not be easily blamed on someone else. Then I was proposed for the Document Engineering team. Now, if I can only keep them from finding out that I'm really just three raccoons in a human suit. Wait, don't print that.

Q What challenges does the FreeBSD Project face with regards to maintaining, and translating, its documentation?

A Sheer mass, for one. There is a huge amount of documentation already, and keeping it up to date is difficult. Systems and programs change constantly, and we have had up to three separate versions of FreeBSD all released and supported at one time.

Another issue is markup languages. We use DocBook, which takes some study to understand and use effectively. We also use mdoc for man pages, which is utterly unlike DocBook. These can both be intimidating for new users. Fortunately, we have the aforementioned huge mass of documents which can be used as examples. The trick is keeping that markup in good condition. It seems like people instinctively choose the worst documents as examples, and copy that bad formatting.

Getting people to review technical documents is complicated by these factors. We are really most concerned with the content being correct, but reading just the content is difficult in a file marked up in DocBook. That is another issue on a long list of things we want to do: use some sort of annotation system to let people read and check the online documents without having to wade through markup.

Q You recently issued a call for testers for translators to provide feedback on using PO/Gettext for documentation translation (<https://lists.freebsd.org/pipermail/freebsd-translators/2015-August/000026.html>). Why

was this toolchain selected and how does it change a translator's workflow?

ATo start, realize that FreeBSD predates such modern technologies as XML. The translations were initially set up many years back, and they worked just like source code. Translation teams would watch for commits to the English documents. Then they would look at the changes and manually translate them into their language. They would have to be careful to preserve the XML markup and even indentation rules for their team, because it all had to be coordinated.

The sheer amount of work and dedication required to keep these translations updated is amazing. Some teams just could not keep up, and we were left without translations to many important world languages.

Even those writing the English documentation had to take special care. Whitespace changes which only affected paragraph rewrapping or tabs or spaces had to be kept separate from content changes so that translators would not needlessly retranslate sections with the same content. Explaining the importance of avoiding whitespace changes to new contributors is surprisingly difficult and confusing. "See this stuff that is invisible? Do not change it, even though it does not matter."

In the meantime, other open source projects started using the gettext "PO" system (https://www.gnu.org/software/gettext/manual/html_node/index.html) for translation. This is the same system used by programmers to translate a program's user interface into other languages. For us, this makes translations much, much easier. Translators do not have to watch for commits to the English version. They do not have to deal with much markup, or worry about integrating changes into

their existing translation. A PO editor (<https://poedit.net/>) just shows them which strings need to be translated. When the English version changes, only the changed sections need new translation, and the PO editor shows how much of the entire document has been translated.

Another big win is "translation memory." Once a string has been translated, the PO editor can remember it and automatically use it when the string appears in a different document. The translator does not have to translate it again, so it reduces their workload.

Members of a translation team can share translation mem-

AIf the documentation is in XML, and there are no existing translations, it could be fairly easy. There are a couple of programs that can extract strings from XML files into PO files and back, like itstool

(<http://itstool.org/>) and po4a (<https://po4a.alioth.debian.org/>).

The basic process is simple: extract the strings from an XML file, use a PO editor to translate them, then build a new XML file with the translated strings. Of course, there are many possible catches along the way.

We have lots of existing translations, and our source English documents are something like 87 megabytes, including the

“THERE ARE A LOT OF PEOPLE OUT THERE WHO WOULD LIKE TO FIX OR UPDATE A SMALL PART OF THE DOCUMENTS, AND WE NEED TO HELP ENABLE THOSE CONTRIBUTIONS.”

—WARREN BLOCK

ory. There are even applications like Pootle (<http://pootle.translatehouse.org/>), which is essentially an online PO editor. Log in to a website and translate documents. Other projects have been doing this for quite a while, but for us it still has that new car smell.

Overall, the PO translation system lets translators concentrate on translating.

QFor readers interested in using PO/Gettext for their own documentation, what is the process for converting existing documentation and integrating this translation method into a build system? What is the size of the existing FreeBSD documentation set and how long did it take to go from documentation conversion, integration, and internal testing to the public call for testing?

FreeBSD.org website. The biggest individual documents are the *FreeBSD Handbook* and the *Porter's Handbook*.

It would be nice to say that existing translations had been converted. However, that has not happened yet. Translations produced with the PO system have been of documents that had not been translated before, or are completely new translations. Some of our translations are so outdated that completely new translations will not be much more work than trying to convert them.

There have been rumors of software that could create the backwards mapping from a translation into a PO file, but it has not yet been located. Investigation is ongoing, and help is always welcome.

QWas the transition to a new translation process a smooth one and were there

any surprises along the way? What sort of feedback have you received from translators?

A Given that the transition has only just started, it is hard to tell yet. The best part is that when people offer to help translate FreeBSD documents, now we can accept that help. They are not required to read and understand the entire FreeBSD Documentation Project Primer and DocBook XML and all the rest.

The best feedback has been “I tried it, and it worked!”

Q Going forward, does the Project have additional plans for easing the learning curve and workload for those interested in contributing to the FreeBSD documentation and its translations?

A The learning curve is a difficult question because it has several parts. Markup is one of those, but we also have style and format rules. Some things can and should be automated. I have some ideas on making DocBook markup easier.

A switch to a different markup system is possible. We have had suggestions that we should switch to Markdown, or mdoc, for everything, or AsciiDoc (<http://asciidoc.org/>). But the current system uses a large toolchain that takes advantage of many of the features of DocBook. Like reimplementing any legacy system, it is not as simple as it might appear at first. Switching would require finding replacements for many of those functions, and some simpler markup systems get that way by sacrificing power or expressiveness.

A lot can be improved with tools to make the current systems easier for end users. A crude example is igor, a program that just looks through documentation and looks for problems. It checks all kinds of things from spelling to format-

ting and style. This takes some of the load off the user, just like a “lint” checker for programming. The current implementation is better than nothing, but there is tremendous potential for improvement in this area.

What if people could use LibreOffice to write documents, producing DocBook output? It is technically possible to configure the “styles” features so there would be styles like “filename”, “command”, and so on. There might be smaller GUI editors that can do the same thing. I’ve only just started looking into this. At this point, it is only an idea.

One problem is that FreeBSD tends to not have lazy contributors. By that, I mean that many of our contributors are so dedicated that they are prepared to go through the pain of manually writing XML markup just to get the thing fixed as soon as possible. In that way, the dedication of our contributors can work against us. Those people can put up with unfriendly processes that might scare off new contributors.

There are a lot of people out there who would like to fix or update a small part of the documents, and we need to help enable those contributions.

Q You are also a member of the DocEng team. What is the role of this team within the FreeBSD Project?

A Mostly, it is to keep the documentation tools working and the documentation functioning. That includes various combinations of all the tools and toolchains for the documents: DocBook, XML, mdoc, XSL, and everything that goes into the website.

The way all these systems interact is intricate and fascinating. Also, at times, horrifying.

Q You recently attended the OpenHelp conference

(<https://conf.openhelp.cc/>) in Cincinnati [Ohio]. Did you have any interesting takeaways from this year’s conference? Why is it important for documentation writers and translators to interact with other open-source projects?

A It is great to interact with the larger communities. For the most part, we share similar problems. Conferences like these provide a chance to see how other groups have gone about solving their problems, and for them to see how well our approaches have worked. Being a part of the larger open-source community is very valuable.

The PO translation system is a good example. In FreeBSD, we had never used it, and so had no knowledge of common problems and solutions with it. Other projects have been using that system for years, and were a valuable source of information and help with it.

This year, there was a great deal of interest in AsciiDoc. This simplified markup system provides enough power for many uses. Like us, other projects struggle with teaching contributors how to use their markup systems, and the question arises of how much complexity is required for the documents they produce.

Interaction with the larger open-source community can help us avoid becoming locked into some dead-end path dependence, and keep us aware of new possibilities. We can learn from them while they learn from us. Everybody wins. Except Cletus. At least the giant squid was happy. •

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.



**Testers, Systems Administrators,
Authors, Advocates, and of course
Programmers** *to join any of our diverse teams.*

**COME JOIN THE
PROJECT THAT MAKES
THE INTERNET GO!**

★ DOWNLOAD OUR SOFTWARE ★

<http://www.freebsd.org/where.html>

★ JOIN OUR MAILING LISTS ★

<http://www.freebsd.org/community/maillinglists.html?>

★ ATTEND A CONFERENCE ★





Events Calendar

The following BSD-related conferences will take place in November. More information about these events, as well as local user group meetings, can be found at www.bsdevents.org.

OpenRheinRuhr • Nov 7 & 8 • Oberhausen, Germany



<http://openrheinruhr.de/> • OpenRheinRuhr is an annual conference on free software, with presentations for both beginners and professionals. There will be at least one FreeBSD presentation, as well as an all-BSD booth in the expo area.

LISA • Nov 8–13 • Washington, DC



<https://www.usenix.org/conference/lisa15> • The annual LISA conference provides a vendor-neutral meeting place for the wider system administration community. There will be a FreeBSD booth in the expo area. A free expo-only pass is available during registration.

Fossetcon • Nov 19–21 • Lake Buena Vista, FL



<http://fossetcon.org/> • Fossetcon is a three-day event focusing on a variety of free and open-source software. Registration, and a nominal fee, is required for this event. There will be at least one FreeBSD presentation, as well as an all-BSD booth in the expo area. The BSDA certification exam will also be available during this event.



FreeBSD Journal's 2014

“BUNDLE” SUBSCRIPTION PACKAGE

Are you missing the 6 back issues from 2014?

Not to worry, the FreeBSD Foundation is pleased to announce a new subscription package that allows you to get **All 2014 Back Issues** for the low price of \$24.99 — that's a \$16.95 savings from the single copy price.

Get the Subscription Set Today

Order as desktop editions at www.freebsdoundation.org or purchase at your favorite app store.





We're excited to announce that March 15, 2015 marks the **15th Anniversary** of the FreeBSD Foundation!

You've helped up accomplish so much during the last 15 years and we look forward to continuing that progress through out the rest of 2015. The areas we'd like to invest in include the following:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure
- And More!

Thank you for all of your continued support. We can't do it without you!



Support FreeBSD®

Donate to the Foundation!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system. Our mission is to continue and increase our support and funding to keep FreeBSD at the forefront of operating system technology.

For 15 years, the FreeBSD Foundation has been proudly supporting the FreeBSD Project and community thanks to people like you. We are incredibly grateful for all the support we receive from you and so many individuals and organizations that value FreeBSD.

Make a gift to support our work in 2015. Help us continue and increase our support of the FreeBSD Project and community worldwide!

Making a donation is quick and easy.
Go to freebsd.foundation.org/donate



The
FreeBSD
FOUNDATION
freebsd.foundation.org